Impact of Transformational Leader Behaviors on Diverse Team Performance and Persistence

by

Jeremy Goodwin

B.S., Aerospace Engineering, Embry-Riddle Aeronautical University (1999)
M.S., Astronautical Engineering, Air Force Institute of Technology (2006)

Submitted to the Department of Engineering and Management
in partial fulfillment of the requirements for the degree of

Master of Science in Engineering and Management

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Engineering and Management
6 August 2021

Certified by .....
Dr. Bryan R. Moser
Academic Director & Senior Lecturer, System Design & Management Program
Thesis Supervisor

Accepted by .....
Joan S. Rubin
Executive Director, System Design & Management Program

Impact of Transformational Leader Behaviors on Diverse Team Performance and Persistence

by
Jeremy Goodwin

Submitted to the Department of Engineering and Management
on 6 August 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Engineering and Management

Abstract

Scholars have studied diversity and diverse teams for more than 60 years. While some studies show that diversity is positively linked to team performance outcomes, many others show the opposite. Indeed, some scholars now refer to diversity as a "double-edged sword" because although it can have its benefits, it also can impose costs on the team. Reagans et al. (2004) suggest that one reason for these conflicting results is a pair of network structural variables in the team: external network range and internal network density. Diverse teams are known for being strong in the former, but weak in the latter. With both variables known to be positively related to team performance, weak internal network density represents an area that leaders can focus on to improve overall performance.

Additionally, an organization's approach to diversity has also been shown to drive diversity-related concerns according to a member's social group representation within a larger group. Specifically, Apfelbaum et al. (2016) studied two approaches that are commonly used across different organizations, and although both are meant to value diversity, they each increase concerns in different social groups and ultimately drive different members away from the organization.

This thesis proposes an experiment that triggers a leader to increase team member support in order to increase internal network density and stave off further costs typically associated with diverse teams. The research presents an agent-based model to simulate multiple team scenarios using both Monte Carlo and single simulations. Though in its early form and needing further empirical validation, the simulations suggest that addressing diversity-related concerns with supportive behaviors can increase internal network density, which in turn can reduce absences and team member turnover. Perhaps more importantly, the model represents a foundation for further study and associated incorporation of additional diversity-related phenomena.

Thesis Supervisor: Dr. Bryan R. Moser
Title: Academic Director & Senior Lecturer, System Design & Management Program

Acknowledgements

"I learned that rewriting a company or university's diversity and inclusion statement matters far less than changing what people believe and how they behave toward one another." - Scott Page (Page, 2017, p.5)

First and foremost, my faith in Jesus was a huge part of this effort. In addition to a Holy stirring I felt in the wake of George Floyd's death on Memorial Day in 2020, the diverse church I attend leaned in to the systemic issues we face as a church and as a society. It helped us grow stronger as a church, and it served as a model for how I led my small but diverse work team through our own challenges. I am grateful for the opportunity to learn from Dr. Mark Batterson and his diverse team of pastors at National Community Church in Washington, DC.

Dr. Ray Reagans, Associate Dean of Diversity, Equity, and Inclusion at the Massachusetts Institute of Technology's Sloan School of Management is the instigator of this thesis. Ray's past research helped me find a starting point for this effort, and his guidance throughout consistently drove me to even better approaches. Further, his suggestions for future enhancements put me in a position where I can take this even further after submission. Even with all these blessings I could never have dreamed to receive, I most treasure our new friendship, and I look forward to working with him further. Thank you, Ray.

Additionally, there are no words to express how grateful I am for Dr. Bryan Moser's support over the last year. In addition to his service to our System Design and Management fellowship cohort, Bryan took me under his wing and helped me do something I never thought I'd be able to do: write a Java-based program. With some programming experience in my own ancient history, Bryan showed me the power of object-oriented programming, and he helped me apply what I was learning in the diversity literature to develop an agent-based simulator. I liken this experience to writing a thesis in a foreign language, and Bryan was always there to help me conjugate new verbs. Bryan, thanks for teaching me this new language, and for all your help getting to where we are today.

Next, there are a number of friends I'd like to thank as well. First, Dr. Robert Introne, Dr. David Long, and Dr. Ellen Pawlikowski were all kind enough to write the recommendations that helped me get into MIT. I am grateful to call them each mentors and friends. Next, Robert "Chad" Holmes is, in my humble opinion, a coding superstar that was always willing to help out a friend in need. And as a newbie to Java, I was definitely a friend in need... a lot. Also, Felicia Burks, Kitara Johnson, Breanna Fulton, Wanda Jones-Heath, Kimberly Pollard, and Linda Arrey are all amazing women of color that I've had the pleasure of meeting and learning from on this journey, and I look forward to their continued friendship. May we never stop learning from each other. Finally, I'm so grateful for Matt and Katie Reilman and Tara Weidus for their friendship and support to me and my family over the last year. Thanks for keeping me sane.

Saving the best for last, I am eternally grateful to my family for their never-ending love and support. I can always count on my grandparents, parents, and each of my 8

siblings to be in my corner. In particular, my Uncle Sean received his PhD in Physical Chemistry from MIT in 1994, and he's a role model that is always encouraging my academic pursuit. Finally, my wife and two sons dealt with life without me being available as much as we would have liked over the last year. Louise, we knew this was not going to be an easy year, but I'm grateful that we always found a way to work through the hard times. And to my boys, one of whom is even now VERY impatiently waiting for me to turn this project in so we can start working on the new Lego Space Shuttle and Hubble Space Telescope together, I'm so grateful I get to be your dad. I love you, I'm so proud of you both, and I can't wait to see what the future holds for you!

Contents

1 Introduction **17**
- 1.1 Research Motivation . . . 18
- 1.2 Thesis Outline . . . 20

2 Literature Review **23**
- 2.1 Teams as Systems . . . 24
  - 2.1.1 Input-Process-Output (I-P-O) Model . . . 24
  - 2.1.2 Input, Mediator, Output, Input (IMOI) Model . . . 25
  - 2.1.3 Model of Overlapping & Co-evolving Team Features . . . 26
- 2.2 Diverse Teams . . . 27
  - 2.2.1 Value of Diverse Teams . . . 28
  - 2.2.2 Performance Mediators of Diversity . . . 29
- 2.3 Leadership of Diverse Teams . . . 33
  - 2.3.1 Team Performance & Perceived Leader Support . . . 33
  - 2.3.2 Leader Support & Transformational Leadership . . . 35
  - 2.3.3 Transformational Leadership & Diverse Teams . . . 36
- 2.4 Chapter Summary . . . 39

3 Research Approach **41**
- 3.1 Research Question . . . 41
- 3.2 Research Hypothesis . . . 42
  - 3.2.1 Combined Integrated Static Model . . . 42
  - 3.2.2 Notional System Dynamics Representation . . . 44
  - 3.2.3 Hypothesis . . . 45
- 3.3 Potential Research Experiment Design . . . 45
  - 3.3.1 Objective . . . 45
  - 3.3.2 Domain . . . 45
  - 3.3.3 Data Collection . . . 46
  - 3.3.4 Execution . . . 47
- 3.4 Simulation Approach . . . 49
  - 3.4.1 Available Modeling Methods . . . 49
  - 3.4.2 Selected Approach: Meso-level Agent-Based Model. . . . 52
- 3.5 Chapter Summary . . . 53

4 Agent-Based Simulation **55**
4.1 Assumptions . . . 55
4.2 Model Introduction . . . 56
4.3 Detailed Model Functionality . . . 58
4.3.1 User Interface (Single Simulation & Monte Carlo Simulation . . . 58
4.3.2 Diversity Model Runner & Diversity Monte Carlo Simulation . . . 62
4.3.3 World . . . 62
4.3.4 Agent . . . 64
4.4 Model Validation . . . 69
4.4.1 Validation #1: Internal Network Density Adjustments . . . 69
4.4.2 Validation #2: Diversity-Concern Generation . . . 73
4.4.3 Validation #3: Random Day Off Generation . . . 74
4.4.4 Validation #4: Agent Departure Generation . . . 75
4.4.5 Validation #5: Transformational Leadership Support Behaviors 76
4.4.6 Validation #6: Sociometric Badge Awareness . . . 78
4.4.7 Validation #7: Monte Carlo Simulation Operation . . . 80
4.5 Chapter Summary . . . 81

5 Discussion of Results **83**
5.1 Base Case Simulation Results . . . 83
5.1.1 Monte Carlo Results . . . 84
5.1.2 Single Simulation Results . . . 92
5.2 Transformational Leadership Simulation Results . . . 93
5.2.1 Monte Carlo Results . . . 94
5.2.2 Single Simulation Results . . . 103
5.3 Sociometric Badge Experiment Simulation Results . . . 105
5.3.1 Monte Carlo Results . . . 105
5.3.2 Single Simulation Results . . . 111
5.4 Significant Result: A Goal to Strive For? . . . 113
5.5 Chapter Summary . . . 114

6 Conclusions, Limitations, and Future Research Suggestions **117**
6.1 Conclusions . . . 117
6.1.1 Summary . . . 117
6.1.2 Experiment Insights & Conclusions . . . 119
6.2 Limitations . . . 120
6.3 Suggestions for Future Research . . . 120
6.4 Closing Remarks . . . 121

A Software Code **123**
A.1 User Interface Controller Files . . . 123
A.1.1 UserInterface.java . . . 123
A.1.2 UserInterfaceInputView.java . . . 124
A.1.3 UserInterfaceResultsView.java . . . 128
A.1.4 inputView.fxml . . . 139

A.1.5 ResultsView.fxml . . . 143
A.1.6 UserInterfaceMonteCarlo.java . . . 148
A.1.7 UserInterfaceInputViewMonteCarlo.java . . . 149
A.1.8 UserInterfaceResultsViewMonteCarlo.java . . . 153
A.1.9 HistoInputs.fxml . . . 169
A.1.10 MonteCarloResultsView.fxml . . . 174
A.2 Environment Files . . . 177
A.2.1 DiversityModelRunner.java . . . 177
A.2.2 DiversityMonteCarloSim.java . . . 184
A.2.3 World.java . . . 197
A.2.4 DiversityApproach.java . . . 225
A.2.5 DemandBase.java . . . 225
A.2.6 DemandFX.java . . . 233
A.2.7 DemandState.java . . . 235
A.2.8 DemandType.java . . . 235
A.2.9 SimResult.java . . . 235
A.2.10 ResultsExporter.java . . . 239
A.3 Agent Files . . . 253
A.3.1 Agent.java . . . 253
A.3.2 AgentMode.java . . . 281
A.3.3 AgentType.java . . . 282
A.3.4 Supply.java . . . 282
A.3.5 SupplyBase.java . . . 284
A.3.6 SupplyFX.java . . . 292
A.3.7 ActFX.java . . . 293
A.3.8 ActType.java . . . 294
A.3.9 Race.java . . . 294
A.3.10 Gender.java . . . 295

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

2-1 Input-Process-Output Model from (Hackman and Morris, 1975) . . . 25
2-2 Input, Mediator, Output, Input (IMOI) Model . . . 26
2-3 Model of Overlapping & Dynamic Team Features (Mathieu et al., 2019) 27
2-4 Transformational Leadership Impact on Team Creativity (Shin and Zhou, 2007) . . . 37
2-5 Transformational Leadership Impact on Team Performance (Kearney and Gebert, 2009) . . . 38

3-1 External Network Range and Internal Network Density Relationship with Performance (Reagans et al., 2004, p.103) . . . 43
3-2 Relationship Among Transformational Leadership and Team Performance (Kearney and Gebert, 2009, p.78) . . . 43
3-3 My Proposed Model Integration . . . 43
3-4 Proposed Systems Dynamics Model . . . 44
3-5 Sociometric Badge (Waber, 2012) . . . 47

4-1 Model Class Map . . . 57
4-2 Single Simulation User Interface . . . 59
4-3 Monte Carlo Simulation User Interface . . . 59
4-4 Validation Test #1a: Homogeneous Team With 0% Interaction *(Value in Equality)* . . . 71
4-5 Validation Test #1b: Homogeneous Team With 100% Interaction *(Value in Difference)* . . . 72
4-6 Validation Test #1c: Heterogeneous Team With 0% Interaction *(Value in Equality)* . . . 72
4-7 Validation Test #1d: Heterogeneous Team With 100% Interaction *(Value in Difference)* . . . 73
4-8 Sample Simulation Output - Validation #5b . . . 78
4-9 Sample Simulation Output - Validation #6a . . . 79
4-10 Monte Carlo Validation Simulation *(Value in Equality)* . . . 81

5-1 Base Case Monte Carlo Results: Internal Network Density & Interactions *(Value in Equality)* . . . 85
5-2 Base Case Monte Carlo Results: Internal Network Density & Interactions *(Value in Difference)* . . . 86

5-3 Base Case Monte Carlo Results: Internal Network Density & Diversity Concerns *(Value in Equality)* . . . 87
5-4 Base Case Monte Carlo Results: Internal Network Density & Diversity Concerns *(Value in Difference)* . . . 87
5-5 Base Case Monte Carlo Results: Diversity Concerns & Agent Absences (*Value in Equality*) . . . 88
5-6 Base Case Monte Carlo Results: Diversity Concerns & Agent Absences *(Value in Difference)* . . . 89
5-7 Base Case Monte Carlo Results: Agent Departures & Agent Absences *(Value in Equality)* . . . 90
5-8 Base Case Monte Carlo Results: Agent Departures & Absences *(Value in Difference)* . . . 90
5-9 Base Case Monte Carlo Results: Project Delays & Agent Departures *(Value in Equality)* . . . 91
5-10 Base Case Monte Carlo Results: Project Delays & Agent Departures *(Value in Difference)* . . . 91
5-11 Base Case Single Simulation *(Value in Equality)* . . . 92
5-12 Base Case Single Simulation *(Value in Difference)* . . . 93
5-13 Transformational Leadership Monte Carlo Simulation Results: Internal Network Density Vs. Interactions *(Value in Equality)* . . . 95
5-14 Transformational Leadership Monte Carlo Simulation Results: Internal Network Density Vs. Interactions *(Value in Equality)* . . . 95
5-15 Transformational Leadership Monte Carlo Simulation Results: Diversity Concerns Vs. Internal Network Density *(Value in Equality)* . . . 96
5-16 Transformational Leadership Monte Carlo Simulation Results: Diversity Concerns Vs. Internal Network Density *(Value in Difference)* . . . 97
5-17 Transformational Leadership Monte Carlo Simulation Results: Absences vs. Diversity Concerns *(Value in Equality)* . . . 98
5-18 Transformational Leadership Monte Carlo Simulation Results: Absences vs. Diversity Concerns *(Value in Difference)* . . . 98
5-19 Transformational Leadership Monte Carlo Simulation Results: Interventions vs. Absence *(Value in Equality)* . . . 99
5-20 Transformational Leadership Monte Carlo Simulation Results: Interventions vs. Absence *(Value in Difference)* . . . 100
5-21 Transformational Leadership Monte Carlo Simulation Results: Support vs. Diversity Concerns *(Value in Equality)* . . . 101
5-22 Transformational Leadership Monte Carlo Simulation Results: Support vs. Diversity Concerns *(Value in Difference)* . . . 101
5-23 Transformational Leadership Monte Carlo Simulation Results: Agent Departures vs. Absences *(Value in Equality)* . . . 102
5-24 Transformational Leadership Monte Carlo Simulation Results: Project Duration vs. Agent Turnover *(Value in Difference)* . . . 103
5-25 Transformational Leadership Scenario Single Simulation *(Value in Equality* . . . 104

5-26 Transformational Leadership Scenario Single Simulation *(Value in Difference)* . . . 105
5-27 Sociometric Badge Awareness Monte Carlo Results: Internal Network Density vs. Interactions *(Value in Equality)* . . . 107
5-28 Sociometric Badge Awareness Monte Carlo Results: Internal Network Density vs. Interactions *(Value in Difference)* . . . 107
5-29 Sociometric Badge Awareness Monte Carlo Results: Diversity Concerns vs. Internal Network Density *(Value in Equality)* . . . 108
5-30 Sociometric Badge Awareness Monte Carlo Results: Diversity Concerns vs. Internal Network Density *(Value in Difference)* . . . 109
5-31 Sociometric Badge Awareness Monte Carlo Results: Daily Leader Supports vs. Diversity Concerns *(Value in Equality)* . . . 110
5-32 Sociometric Badge Awareness Monte Carlo Results: Project Duration vs. Absences *(Value in Difference)* . . . 110
5-33 Sociometric Badge Awareness Experiment Single Simulation *(Value in Equality)* . . . 111
5-34 Sociometric Badge Awareness Experiment Single Simulation *(Value in Difference)* . . . 112
5-35 Caption . . . 113
5-36 Transformational Leadership Scenario: Absence vs. Internal Network Density & Leader Effectiveness *(Value in Equality)* . . . 114

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

4.1 Validation Test #1 Results 71
4.2 Validation Test #2a Results *(Value in Equality)* 74
4.3 Validation Test #2b Results *(Value in Difference)* 74
4.4 Validation Test #3a Results *(Value in Equality)* 75
4.5 Validation Test #3b Results *(Value in Difference)* 75
4.6 Validation Test #4a Results *(Value in Equality)* 76
4.7 Validation Test #4b Results *(Value in Equality)* 76
4.8 Validation Test #5a Results *(Value in Equality)* 77
4.9 Validation Test #5b Results *(Value in Difference)* 77
4.10 Validation Test #6a Results *(Value in Equality)* 79
4.11 Validation Test #6b Results *(Value in Difference)* 79
4.12 Validation Test #7a Results *(Value in Equality)* 80

5.1 Correlation Matrix: Base Case Monte Carlo Simulation *(Value in Equality)* 84
5.2 Correlation Matrix: Base Case Monte Carlo Simulation *(Value in Difference)* 84
5.3 Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Equality)* 94
5.4 Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Equality)* 106
5.5 Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Difference)* 106

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

> *"We should reflect on how seeking evidence of a business case for diversity reifies the status quo and legitimates the idea that some people belong and deserve to be included, while other people have to go above and beyond to prove their worth. The questions we ask about diversity have power. They have underlying assumptions that often go unspoken." - Katherine Phillips (Page, 2017)*

I have two sons, the oldest of which has significant physical and mental special needs. He is eight years old, cannot talk, has trouble walking, and does not understand social morays. As much of a struggle as raising him can be at times, being his (and his brother's) dad is the most rewarding experience of my life. Important to this thesis, he's given me a renewed compassion for people and their stories.

One story in particular motivated this work, and it came on Memorial Day in 2020, just months after my acceptance to the System Design & Management (SDM) program at Massachusetts Institute of Technology (MIT). Watching the video of George Floyd, a Black man, calling out for his mother as a White police officer knelt on his neck for over nine minutes triggered new thoughts and emotions associated with my own children. In particular, I had never worried about my kids being killed because of some unwarranted prejudice against them. It occurred to me that my oldest's special needs could change that, and I wondered what it would be like if it was him on the ground.

Professionally, the aftermath of George Floyd's death was also very powerful. I led at the time a small but diverse team that had strong bonds already, but because we had never talked about any diversity-related issues as a team before, some began to surface. Thankfully, we felt safe together and talked about them with each other, and we grew even stronger as a team.

In sum, George Floyd gifted me with a better understanding of my own underlying fears, with an opportunity to practice leadership in a meaningful way, and with a new motivation to understand and help carry burdens of those I get to serve as a leader. This thesis marks my professional start of using those gifts.

1.1 Research Motivation

The SDM program at MIT is designed to "educate leaders to solve the world's complex problems," and it is a fellowship of mid-career leaders that want to "lead engineering rather than leave it"[1]. Further, it is particularly strong at designing (and decomposing) complex sociotechnical systems in which both humans and technology interact to produce the system's outcomes. That is, it recognizes that a system is composed not only of its hardware and software, but also of the people and the processes they follow to bring a system to fruition.

As I prepared to start the program, I resolved to use the incredible opportunity to better understand the systems associated with diverse teams in the workplace so that members of the teams I lead will know that they belong. To this end, there are many different types of diversity and diversity-related concerns, and it is therefore important to first define what these terms mean in the context of this thesis. Rather than adding yet another definition to the research body, I particularly appreciate and will use the definition Williams' (also referred to herein as Katherine W. Phillips) and O'Reilly developed during their review of 40 years of diversity-related research:

- **Diversity -** "Any attribute people use to tell themselves that another person is different." (Williams and O'Reilly III, 1998)
- **Diversity concern -** Concerns that group members have related to the "negative implications of being distinct or 'standing out' because of their group membership."(Apfelbaum et al., 2016, p.2). As a result of interactions with other group members, these concerns can lead underrepresented team members to feel excess scrutiny or stereotyping (Kanter, 1977).

Importantly, Williams and O'Reilly go on to note that a person's more visible characteristics such as race, sex, and age are likely to drive more salient in- and out-group categorization than other characteristics like education and experience (Williams and O'Reilly III, 1998).

Additionally, there are at least two main categories of diversity found in research:

- Identity (also referred to as surface-level) diversity (i.e. differences in race, gender, age, tenure, sexual orientation, etc.)
- Cognitive (or deep-level) diversity - differences in education, experience, etc. Also categorized as differences in attitudes, beliefs, and values (Williams and O'Reilly III, 1998)

Though I am specifically interested in and focused herein on identifying and addressing concerns associated with identity diversity, I do not refrain from discussing cognitive diversity as it is often a team characteristic based on identity diversity as well.

Additionally, as alluded to earlier, I am also interested in how leadership fits in and can impact teams with diversity concerns. As the chapter's opening quote suggests,

[1]SDM Homepage, www.sdm.mit.edu

this work does not "seek evidence of a business case for diversity," but instead it leverages decades of diversity- and leadership-related research to present a case for building *leaders* of diverse organizations. My hope is that this work will help leaders find ways to help more people feel like they belong such that they perform better and stay with their organizations longer.

I discuss my literature review in Chapter 2, but there are three previous research efforts that should be mentioned upfront for the particularly strong role they played in motivating my work. First and foremost is a series of studies done by Dr. Evan Apfelbaum (MIT), Dr. Nicole Stephens (Northwestern University), and Dr. Ray Reagans (MIT).

The authors collaborated on research that ultimately revealed differing diversity-related concerns in social groups according to their level of representation within a larger group. In their paper entitled, "Beyond One Size Fits All: Tailoring Diversity Approaches to the Representation of Social Groups," the authors report that an organization can moderate diversity-related concerns by the approach they take to diversity, and that by doing so, they may find their employees perform better and stay with the organization longer (Apfelbaum et al., 2016). Specifically, they found that members of social groups with *low representation* in the larger group have less concerns when an organization treats everyone equally and gives everyone equal opportunity (*"value in equality"*). On the other hand, they also found that members of social groups with *moderate representation* have less concerns when they feel the organization values them for the unique skill sets they can offer (*"value in difference"*). The authors close this particular effort by coding diversity policy statements from 151 United States law firms and indeed found that lower attrition rates were achieved among the different social groups according to each firm's diversity policy statements.

Next, another diversity-related effort that had a significant role in motivating this work was conducted by Anne Tsui and Terri Egan (both of the University of California - Irvine) and Charles O'Reilly (University of California-Berkeley). Their paper, "Being Different: Relational Demography and Organizational Attachment," was published in 1992 in the Administrative Science Quarterly journal and helped steer my literature review. Their work empirically investigated how demographic variables impact measures of organizational attachment such as absenteeism, psychological commitment to the organization, and an employee's intent to stay with the organization. In sum, they found that an *increase* in team diversity was associated with *lower* levels of psychological attachment, thus leading to *increased* turnover (Tsui et al., 1992).

Finally, the last primary research effort that motivated this work studied leadership behaviors that led to an employee's perception, or lack thereof, of leader support. Dr. Teresa Amabile at the Harvard Business School (HBS) led the work and published the results in 2004 in The Leadership Quarterly with Elizabeth Schatzel (HBS), Giovanni Moneta (HBS), and Steven Kramer (Wayland, MA) (Amabile et al., 2004). The effort built on Dr. Amabile's "componential theory of creativity," which suggests leaders can support their teams through "both instrumental (or task-oriented) and socioemotional (or relationship-oriented) actions" on a daily basis, thus ultimately leading to enhanced creativity and team performance (Amabile et al., 2004,

p7). Moreover, because one of the typical benefits of diverse organizations is their ability to be more creative than homogeneous organizations (Page, 2017), it stands to reason that leader behaviors geared towards providing this support may also lead towards increased organizational attachment by members of underrepresented social groups. Finally, as Shin and Zhou (2007) show, transformational leadership positively moderates diversity and team creativity. Therefore, it also stands to reason that transformational leadership behaviors may also positively moderate diversity towards team performance.

As I reflected on these research efforts, it occurred to me that we may be able to apply SDM principles, methods, and tools to peer deeper into diversity teams as sociotechnical systems. To keep the scope of this thesis manageable, I focus on one tool that seems particularly adept at offering a strong framework to build upon this past research: the agent-based model (ABM).

ABMs can predict programmatic outcomes (e.g., cost, schedule, quality) as a result of typically unconsidered factors such as team coordination distance, design complexity and skill set, and required concurrent design efforts. That is, they use a bottoms-up approach to model how people/teams/organizations work in coordination with other people/teams/organizations according to pre-defined rules like task prioritization and work schedules to predict a program's outcomes. Due to a project's activities that require extensive coordination, wait time, and rework that are hard/impossible to predict when defining top-down plans, these models can predict significantly different outcomes than what are originally expected.

By leveraging past research results, we can use ABMs to predict programmatic outcomes as a result of diversity-related concerns and the behaviors leaders choose to mitigate them. It should be noted, however, that ABMs and the insights they generate can include unconscious bias. To mitigate this risk, I started this work with the following controls in mind:

1. Develop the software code from scratch to eliminate outside bias
2. Clearly acknowledge all assumptions
3. Develop software based on peer-reviewed research
4. Provide the software code in this thesis for transparency

1.2 Thesis Outline

Chapter 1 introduces the thesis topic and my motivation for research in this area.

Chapter 2 provides a review of research on the systemic aspects of teams, on diverse teams, and on leadership of diverse teams.

Chapter 3 leverages the aforementioned literature review to introduce my research question and hypothesis. The chapter then presents a potential experiment to address my research question, which was not possible in this time frame due to Corona Virus-19 pandemic restrictions. In place of an actual experiment, the chapter introduces

various approaches to simulate the experiment, and then closes with rationale for deciding to develop a custom agent-based model (ABM).

Chapter 4 discusses the ABM that I, with input from others noted in my Acknowledgements, developed to begin addressing the thesis research questions. Importantly, it documents the necessary assumptions for this first iteration of the model.

Chapter 5 documents the model results and, given the assumptions documented in Chapter 4, identifies insights for further consideration.

Chapter 6 closes the thesis with my research conclusions. The chapter also acknowledges the limitations of the model and presents potential areas for further research.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Literature Review

> *"It is hardly possible to overrate the value... of placing human beings in contact with persons dissimilar to themselves, and with modes of thought and action unlike those with which they are familiar.... Such communication has always been, and is peculiarly in the present age, one of the primary sources of progress." - John Stuart Mill, Principles of Political Economy, 1848*

According to a 2020 study by the Society of Human Resource Management, there is a significant difference among minority perceptions of inequity in racially diverse teams and that of what the majority perceives (SHRM, 2020). The study goes on to show that this difference in perception gives rise to diversity concerns regarding discrimination and incivility.

While this thesis is not limited to an investigation of race diversity, the above study's findings underpin the generation of concerns that ultimately affect the thesis topic of diverse team performance. Specifically, as diversity concerns rise, workplace performance diminishes and workers then leave their organizations (Apfelbaum et al., 2016). Not only do organizations then lose the investments they've made in these individuals, but they also lose the opportunity to capitalize on what Page calls the "bonus" that the respective diverse teams may have offered (Page, 2017).

Therefore, because individuals are the building blocks of teams, and because teams are now the building blocks of many of today's organizations (Mathieu et al., 2018), it is important to understand where these diversity concerns come from and how they might be mitigated. I suggest the recent conceptualization of teams as dynamic and complex systems that evolve (Kozlowski and Chao, 2018; Mathieu et al., 2019) is the right place to start. To develop how diverse teams and their leadership might be conceptualized as parts of dynamic and complex systems, I offer the following literature review organized as follows:

- Teams as systems - a brief introduction into the systemic nature of teams
- Diverse teams - a review of diversity research in the context of teams as systems, including a discussion on key factors that mediate their performance

- Leadership of diverse teams - specific focus on leader behaviors that impact the performance of diverse teams

2.1 Teams as Systems

As defined by Kozlowski & Ilgen, teams are made up of:

> "two or more individuals who socially interact (face-to-face or, increasingly, virtually); possess one or more common goals; are brought together to perform organizationally relevant tasks; exhibit interdependencies with respect to workflow, goals and outcomes; have different roles and responsibilities; and are together embedded in an encompassing organizational system, with boundaries and linkages to the broader system context and task environment (Kozlowski and Ilgen, 2006, p. 79)"

There is no shortage of team-based research, nor of reviews of team-based research. McGrath et al. (2000) and Mathieu et al. (2018), for example, offer exceptional summaries of the research conducted in the 20th century. Other notable reviews include Levine and Moreland (1990), Cohen and Bailey (1997), Ilgen et al. (2005), and more recently, Mathieu et al. (2017). With such a plethora of available outstanding research reviews, a full review of the literature here would be redundant.

Instead, this section offers a brief introduction to past conceptualizations of teams as systems, beginning with the Input-Process-Output (I-P-O) model. It then discusses an evolution to the Input, Mediator, Output, Input (IMOI) model to address adaptation, dynamics and complexity, followed by a more recent model highlighting the co-evolving and overlapping nature of team features. This last model discusses recent shifts away from what could be construed as the more linear I-P-O and IMOI models to more of a relational and dynamic conceptualization over the last decade (Kozlowski and Chao, 2018).

2.1.1 Input-Process-Output (I-P-O) Model

First introduced by McGrath (1964) and then refined by Hackman and Morris (1975), the I-P-O model illustrates how teams can be conceptualized as systems by their group interaction processes that interact on multiple types of inputs to produce outcomes. As Figure 2-1 illustrates, the inputs could be from an individual (e.g., experience, personality), group (e.g., task structure, group size, cohesiveness), or the environment (e.g., reward structure, environmental stress). Then, as a result of these interaction processes, the outputs are characterized as either performance-related (e.g., quality, speed, etc.) or "other" (e.g., member satisfaction). Researchers essentially converged on using this model (Ilgen et al., 2005), and it therefore offered a lasting framework for much of the early research (see Steiner (1972), McGrath (1984), or Hackman (1987) for examples).

Mathieu et al. (2008) confirm the usefulness of the model over a 40+-year review and noted that it had been modified many times along the way. Reasons for its

modifications included the need to put teams in larger contexts, include temporal elements, or cover something that had otherwise gone overlooked (Mathieu et al., 2008). More recently, Kozlowski and Chao (2018) also noted value in the I-P-O model, but posit that its static nature - and the static nature of the research it underpins - limits its application to dynamic and complex systems.

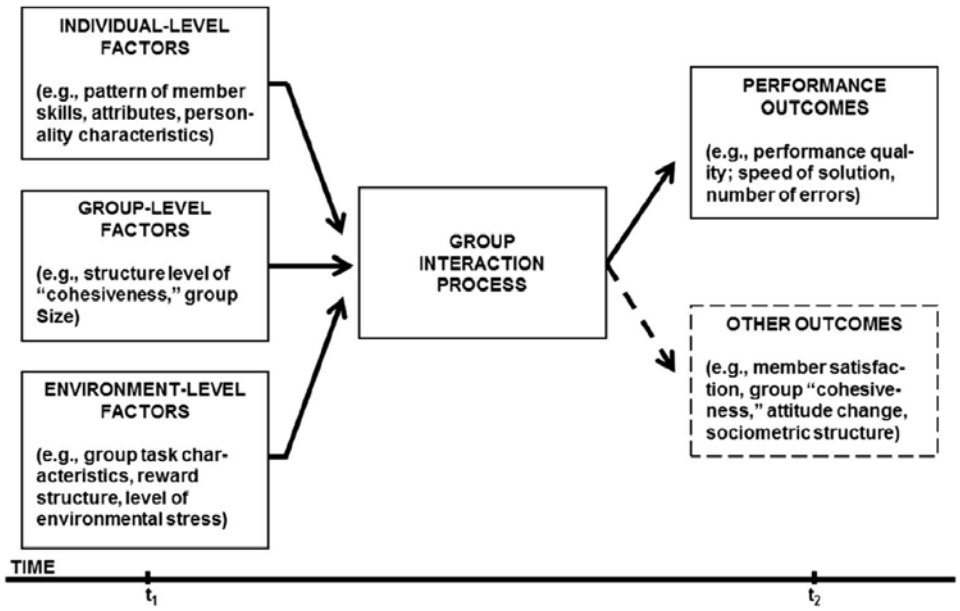

Figure 2-1: Input-Process-Output Model from (Hackman and Morris, 1975)
Reprinted from Mathieu et al. (2018)

2.1.2 Input, Mediator, Output, Input (IMOI) Model

As the research progressed into the more dynamic, adaptive, and complex nature of teams as systems, some scholars moved to an evolved version of the I-P-O model known as the IMOI model introduced by Ilgen et al. (2005). According to Ilgen et al. (2005), the IMOI model was needed for at least three reasons:

- A recognition that many of the mediating factors between inputs and outputs were not "processes"
- A recognition that there is no explicit feedback mechanism in the I-P-O construct (though the early creators of the model do stipulate room for feedback loops; see Hackman (1987) and McGrath et al. (2000) for examples)
- A recognition of the I-P-O model's linear nature that struggled with modeling how processes might interact with other processes, or how emergent states that develop over the life of the team might be considered.

In the new IMOI model, "mediators" replace "processes" to address the first deficiency, an "input" is placed at the end to address the second deficiency with an eye

toward feedback for the next iteration through the model, and the "dashes" from I-P-O were removed to reduce the implication that the model is linear in nature.

Importantly, as shown in Figure 2-2, this model affords the advantage of explicitly considering emergent states such as psychological safety and collective affect. It is these emergent states that Marks et al. (2001) noted to be mediators outside the of traditional "process" construct, which partially led to Ilgen et al. (2005) developing the IMOI construct.

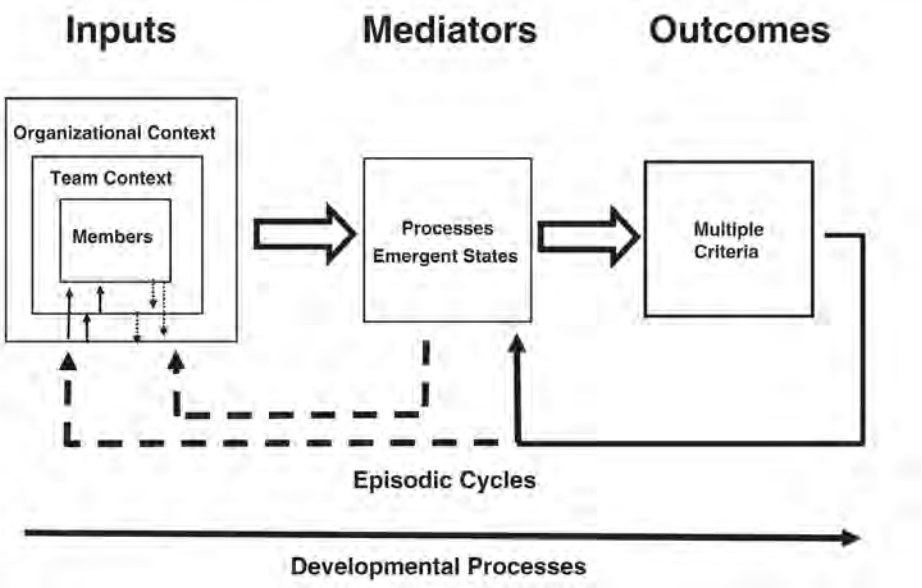

Figure 2-2: Input, Mediator, Output, Input (IMOI) Model
(reprinted from Mathieu et al. (2018) with credit to Ilgen et al. (2005) for coining the IMOI term)

2.1.3 Model of Overlapping & Co-evolving Team Features

Mathieu et al. (2017) acknowledged the usefulness of both the I-P-O and IMOI models as unifying research frameworks, and then offered a new perspective that attempts to capture the overlapping nature of team features (see Figure 2-3). The model builds on the IMOI model by capturing structural (Region A) and compositional (Region B) features as inputs, the mediating mechanisms (Region C), and the outcomes in the center of the diagram. Importantly, Regions D, E, and F of this model now more explicitly identify systemic variables that are recurring, time-dependent, and/or at the intersection of traditional IMOI boundaries (Mathieu et al., 2017).

An emerging theme of this latest research is that today's technology (e.g., machine learning, speech characterization, facial recognition software, etc.) with its capability to collect vast amounts of data, can offer incredible insight into the dynamics of team research (Kozlowski and Chao, 2018; Mathieu et al., 2019).

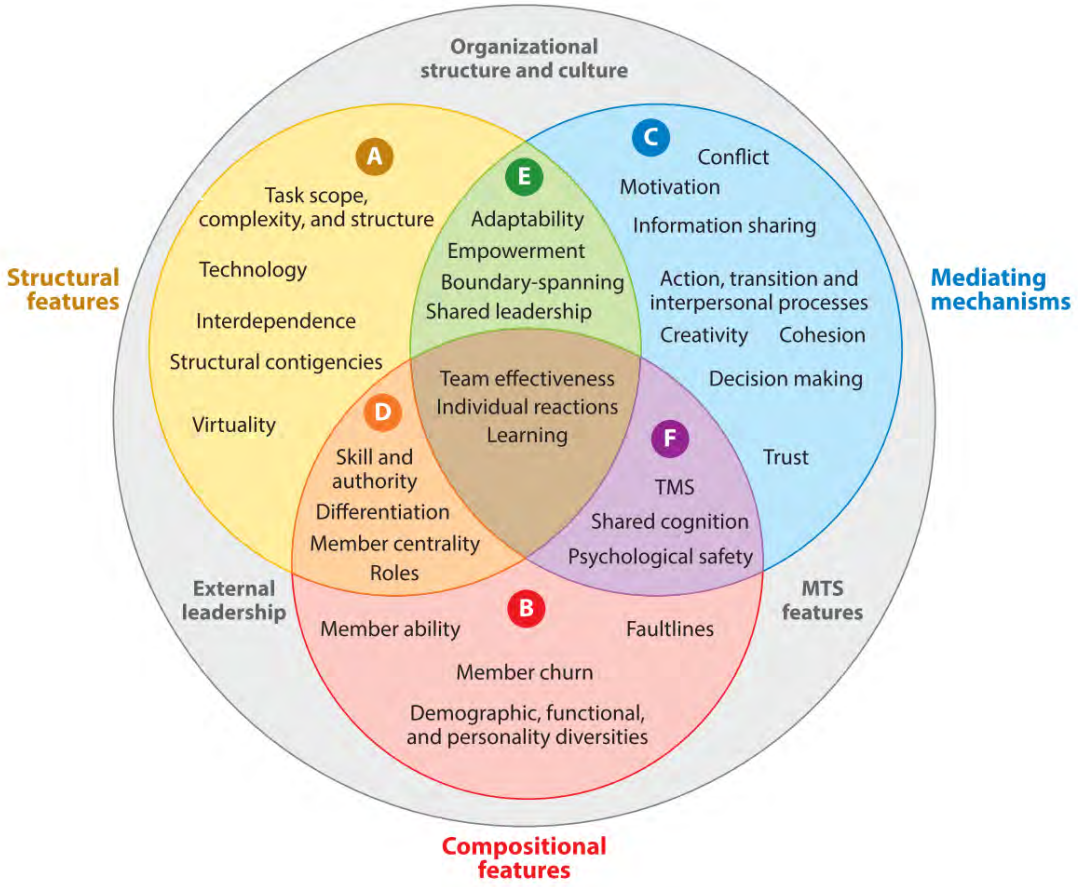

Figure 2-3: Model of Overlapping & Dynamic Team Features (Mathieu et al., 2019)
(Reprinted with APA permission from "A Century of Work Teams in the Journal of Applied Psychology." by J.E. Mathieu, J.R. Hollenbeck, D. van Knippenberg, and D.R. Ilgen, 2017.)

2.2 Diverse Teams

In the SDM program we define a system's "value" as the benefit it brings to its stakeholders at the cost in which those stakeholders have to pay. Similarly, diverse teams can be said to have value in that they can bring benefit to an organization, but they also can incur costs. To this end, many researchers refer to diversity as a "double-edged sword" (see Carter and Phillips, 2017 and Milliken and Martins, 1996 for examples).

As with the team-based literature, a full review of diversity-based research is also beyond the scope of this thesis (see Williams and O'Reilly III, 1998 or De Abreu Dos Reis et al., 2007 for in-depth reviews). Instead, this section first discusses the value (benefits and costs) of diversity, then highlights key insights into the mediators between diversity and team performance. The section concludes with a brief collection of the research's recommendations to leaders of diverse teams.

2.2.1 Value of Diverse Teams

Scholars have been investigating whether or not the benefit of diverse teams is worth the cost for decades. Therefore, as with the amount of team-based research, there is a significant amount of research in this area as well. Unfortunately, the results are divided and often confusing. The research relies on inconsistent theories (Williams and O'Reilly III, 1998) and definitions of diversity (Bell et al., 2011), and appear to drive mixed research results that are often inconsistent with each other. As just one example, (Horwitz and Horwitz, 2007) found in a review of 1000+ studies that identity diversity "exhibited virtually no relationship with the quality of team performance (p.1001)," whereas many other reviews like the more specific 30-year research review by Bell et al. (2011) found a negative correlation between race, gender and team performance.

On one hand, with the above acknowledgement in mind, some research suggests that diverse teams can benefit jobs that involve creativity, innovation, problem solving, making predictions, and discerning truth (Page, 2017). While this was especially prevalent in careful laboratory studies, some scholars also found it true in some real-world empirical cases as well (Williams and O'Reilly III, 1998). As one example, O'Reilly and colleagues found that racial diversity among 32 teams in a large firm correlated positively with creativity and implementation (O'Reilly III et al., 1998).

Additionally, Page also notes a specific practice where firms will leverage individuals from underrepresented groups on tasks associated with customers that reflect their shared identities (Page, 2017). That is, for companies desiring to increase their market share among women, successful companies incorporate women on the team responsible for making it happen. However, Thomas and Ely (1996) note that many organizations too often relegate members of these underrepresented social groups to only these types of identity-sharing tasks, therefore missing out on the other benefits they can provide.

Moreover, some research even suggests simply by having diverse members on a team, majority members may change their behavior and produce positive results (e.g., Sommers (2006)). In a study where diverse group members did not introduce any new ideas to the group's discussions, results showed that the social concerns associated with their presence still stimulated better overall cognitive gains within the group (Phillips et al., 2009). Williams and O'Reilly note that these types of findings are consistent with information and decision-making theory in that access to diverse task-relevant information can increase a group's information and associated problem-solving perspectives (Williams and O'Reilly III, 1998).

On the other hand, the vast majority of empirical evidence indicates diversity impede group performance (Williams and O'Reilly III, 1998). Typical outcomes include higher absenteeism and turnover (Tsui et al., 1992) and decreased job satisfaction Riordan and Shore (1997) as team diversity increases. Similarly, (O'Reilly III et al., 1998) found that heterogeneous groups have lower job satisfaction and higher turnover than more homogeneous groups. There are also negative outcomes associated with diversity between leader and subordinates that include increased ambiguity in roles, lower performance evaluations, and low levels of attraction (Tsui and O'reilly, 1989).

These results are consistent with the social categorization theory and similarity/attraction theory that underpin the majority of their studies (Williams and O'Reilly III, 1998), which generally suggest that as individuals self-categorize into subgroups, are placed into subgroups, or work with others dissimilar to them, performance results typically are negatively impacted.

Further, these negative outcomes of diverse teams are not limited to the underrepresented team members. Majority members also can feel lower job satisfaction associated with increased conflict, thus also leading to their increased absenteeism and turnover (Tsui et al., 1992) as well. Moreover, these results may also be nonsymmetrical (Wharton and Baron, 1987). Indeed, Tsui et al. (1992) found instances when men felt less attached to groups of primarily women than the attachment women felt in groups when men were in the majority.

Along these lines and building on the (Stephan and Stephan, 1985, p.163) finding that "people who regard themselves as superior experience anxiety concerning interaction with others who are regarded as inferior," Tsui et al. (1992) notes that team members in the majority may feel a threatened self-esteem that leads to less interaction, more stereotyping, and more assumed dissimilarity with the out-group members. Accordingly, Tsui et al. (1992) go on to note that majority members may be strongly incentivized to self-categorize into groups they perceive will maintain positive self identities.

Given this demonstrated complexity and ambiguity in the diversity research, Reagans and Zuckerman offer network-based insight that may help explain some of the results to date. Specifically, they found that increased diversity can enhance a team's access to external networks, but can also diminish the team's internal network density (Reagans et al., 2004). Since external network range and internal network density are both positively correlated in research with overall team performance, the rationale for getting varied results throughout the history of diversity-based research becomes more understandable (Reagans et al., 2004):

> "The fundamental difficulty with both research and policies that focus on the effect of demographic diversity on performance derives from the difficulty of predicting which pathway [external network range or internal network density] will be stronger than the other, because there is little theoretical or empirical basis for the prediction." (Reagans et al., 2004, p. 126)

In sum, the value of diverse teams is quite complex. They can clearly bring value to an organization, but they can also expose the organization to higher costs for all team members. Some of the mediating factors on this performance are discussed in the next section.

2.2.2 Performance Mediators of Diversity

Leveraging mediating factors found by (Williams and O'Reilly III, 1998) and (Mathieu et al., 2017), this section discusses research associated with the significant mediating factors of diversity towards team performance.

Common Goals and Identity. As early as 1936, Sherif wrote that in- and out-group bias can be mitigated by giving team members common goals and identities to promote solidarity (Sherif, 1936). According to similarity/attraction theory, one of the three main theories that underpin diversity research, common goals and identities build similarity and attraction among the team members, thus enabling more desirable interaction than that of dissimilar groups (Williams and O'Reilly III, 1998).

A number of recent studies regarding mediators in this topic are directly applicable to diversity. In one such example, Gaertner and colleagues conducted a study that varied whether or not groups perceived themselves as part of a larger group, as well as whether or not group interaction was present Gaertner et al. (1990). Their results confirmed a reduction in bias both when members of subgroups perceive themselves as part of a larger group, as well as when members of the subgroups were required to interact together (Gaertner et al., 1990).

Similarly, Mullen and Cooper also studied cohesiveness as it relates to performance and found that the link is not only highly significant, but that it gets strongest among small, empirical (vs. lab) groups (Mullen and Copper, 1994). Moreover, the authors found that the strength of the relationship was more a result of shared commitment to task rather than personal attraction or pride associated with the group (Mullen and Copper, 1994).

Finally, Phillips and colleagues found that identity-diverse teams had improved problem-solving results when members of the majority shared some level of agreement with an underrepresented member (Phillips et al., 2009). They found that majority members feared losing attachment with other majority members as a result of aligning with a different sub-group, and this fear was motivation to reconcile the differences, which in turn led to better outcomes (Phillips et al., 2009).

On the contrary, however, when team members do not share common goals or identity, there is increased evidence for dysfunction among diverse groups, including "group process and performance loss, including less positive attitudes, less frequent communication, and a higher likelihood of turnover from the group, especially among those who are most different" (Williams and O'Reilly III, 1998, p.11). Social categorization theory, another main theory that frames diversity research (Williams and O'Reilly III, 1998), suggests that one of the reasons diverse teams have trouble is because members will self-categorize with groups they perceive as more positive (Kramer, 1991). As this categorization occurs, people can lose their shared identity as groups experience the aforementioned negative outcomes. For example, Pfeffer (1985) found that team diversity predicted more turnover among heterogeneous groups at the corporation level, and more turnover among those least similar at the individual level. Tsui et al. (1992) also found that demographic differences led to higher turnover and absenteeism.

Task Interdependence. There appears to be a strong positive relationship between task interdependence and team performance (Horwitz and Horwitz, 2007) that may be due to its ability to positively mediate intrateam trust on team performance (De Jong et al., 2015). Similarly, in a mixed result study of 45 teams across five

technology companies, Ancona and Caldwell also found that the nature of the task and group performance were positively correlated with tenure diversity if tasks were more complex (Ancona and Caldwell, 1992). However, though the diversity did have a positive effect on performance, the increase was not enough to prevent the overall performance from being negative (Ancona and Caldwell, 1992).

Pelled et al. (1999) also found mixed results in their study of diversity and conflict in that task interdependence generated task conflict that the team perceived as positive, thus serving as a positive mediator between functional diversity and team performance. However, that task interdependence was not enough to overcome the emotional conflict generated as a result of race and tenure diversity (Pelled et al., 1999). Further, the authors were surprised to find that there was no significant relationship between emotional conflict and team performance, noting that the teams they studied must have found a way to work around it (Pelled et al., 1999).

While it does not appear that task interdependence alone can mitigate the challenges associated with the complexity of diverse teams, these two (and other) studies indicate that it may be an important mediator towards increased performance.

Collective Culture. O'Reilly and Chatman in their review of organizational culture research conclude that culture is a "prevalent social control system" (p.187) and that collectivistic cultures can moderate the negative effects of social categorization (O'Reilly and Chatman, 1996). Similarly, groups with greater collectivistic cultures have been shown to perceive the organizational conflict associated with their dissimilarity as beneficial to their performance outcomes (Chatman et al., 1998). With respect to diversity and in line with the "Common Goals and Identity" research presented above, Chatman et al. (1998) noted specifically that diversity benefits are more likely to manifest when members organically help to establish cultures from the bottom up that prioritize membership and interests of the organization as a whole over individualistic interests.

Along these lines, Nishii (2013) found that inclusive cultures can moderate the negative consequences of diverse teams. Research from Page (2017) confirms this finding with results from technical organizations that were able to increase their "diversity bonus" because they cultivated inclusive cultures with better listening team members that were more open to others' ideas.

Further, Cox et al. (1991) offers an example in which a Prisoner's Dilemma study revealed that individuals from more collectivistic cultures (i.e., Asian, Hispanic, and Black individuals were much more cooperative than those from individualistic cultures (i.e. Anglo-Americans) (Earley, 1991). While this relationship is not always true (see Espinoza and Garza (1985) for an example of when both Anglo- and Hispanic-Americans were cooperative if their group was in the majority), it is an important finding because it illustrates that the actual proportions of subgroups within the group at large is important to team outcomes (Williams and O'Reilly III, 1998). Indeed, the work by Apfelbaum et al. (2016) that sparked the present research effort confirms the importance of proportionality as well.

Trust. In a review of 112 independent studies of 7,763 teams, De Jong et al. (2015) found that intrateam trust consistently has a strong impact on team performance, even after controlling for leadership and past team performance. Two of the three important moderators they found in this relationship are related to strengths associated with strong diverse teams: task interdependence and skill diversity De Jong et al. (2015). That is, while the study did not necessarily focus on diversity, its findings appear important to the topic.

"Few things are faster at killing a shift to a new way of thinking about diversity than feelings of broken trust" (Thomas and Ely, 1996, p.90). O'Reilly III et al. (1993) found that greater tenure diversity leads to decreased trust, and Brewer (1999) found that cooperation among diverse teams has a negative correlation with performance when trust is not present. Further, conflict among team members lacking trust is likely to be more emotionally charged, whereas conflict among trusting teams tend to be more intellectual in nature (Nishii, 2013).

In sum, research shows trust to be a critical factor in performance, but it may be harder to come by in diverse teams.

Time. Research indicates time can be a powerful moderator of conflict amongst diverse teams (Pelled et al., 1999). While acknowledging specifically that racial diversity can lead to increased emotional conflict, Pelled et al. (1999) found that this conflict can diminish as time passes while team members better align with the task at hand (Harrison et al., 1998). For this reason, short-term studies claiming "good" or "bad" outcomes of diversity likely are premature (Williams and O'Reilly III (1998), Bell et al. (2011)).

Watson and colleagues offer a notable example of this phenomenon. The researchers conducted a 17-week study that found homogeneous teams initially performed better than heterogeneous teams (Watson et al., 1993). The research went on to show, however, that while both groups did improve over time, the heterogeneous team caught up with and ultimately passed the homogeneous team in terms of the range of perspectives and number of alternative solutions they identified (Watson et al., 1993). While range of perspectives and number of alternative solutions were only 2 of the 6 variables under investigation, the ability of the heterogeneous team to "catch up" regarding the other 4 variables leads to a question regarding whether more time would have enabled more surpassed variables.

Tolerance. Hoffman et al. (1962) showed that a culture of tolerance for other team members can also lead team members to work through the conflict of differing opinions and perspectives. Specifically, the tolerance individuals had for one another helped subordinates share their ideas because they were not afraid to disagree with those in authority, especially when those in authority heard their subordinate inputs first before sharing their own (Hoffman et al., 1962).

2.3 Leadership of Diverse Teams

Stogdill (1974) presents a comprehensive literature review on leadership and concludes that there is no agreed-upon definition of *leadership*. That is, though many definitions involve some form of influence, there is inconsistency in terms of how the definitions address (or do not address) individual traits, behaviors, functions, interactions, relationships, etc. (Yukl and Van Fleet, 1992; McGrath, 1962). Leadership has also been defined in terms of of the formality (i.e. formal vs. informal) of the role as well as the position of the leader (internal vs. external) with respect to the team (Morgeson et al., 2010). This thesis adopts the functional leadership theory definition of a leader's role offered by McGrath (1962) as:

> **Leader role -** "to do, or get done, whatever is not being adequately handled for group needs" (McGrath, 1962, p.5).

Further, while acknowledging that leadership functions can be performed by both internal or external leaders (Morgeson et al., 2010), it is the local leaders that are likely to be one of the most significant factors driving an employee's daily work experience (Amabile et al., 2004). Moreover, according to the Amabile (1988) *componential theory of organizational creativity*, these local leaders drive the perceived work environment, which can significantly affect both individual and team creativity. Therefore, these local leaders and their behaviors represent an important area of study.

For example, some of the more positively influencing behaviors local leaders can perform include directly helping with projects, developing their subordinates' expertise, and enhancing their subordinates' intrinsic motivation (Amabile et al., 2004). Other successful behaviors include supporting the team within the organization, valuing team member contributions, giving constructive feedback, and being willing to listen to new ideas (Amabile, 1988). These behaviors lead to greater individual creativity, which in turn leads to greater organizational success (Amabile, 1988; Oldham and Cummings, 1996).

Finally, as will be shown, these types of behaviors align with a style of leadership known as **transformational leadership** in which leaders leverage a strong moral foundation to lead using behaviors in these categories: *idealized influence*, *inspirational motivation*, *individual consideration*, and *intellectual stimulation* (Burns, 1978). Accordingly, this section begins first with a discussion of the significant behaviors that drive perceptions of leader support (and ultimately, performance), followed by a brief description of behaviors that transformational leaders use. The section then concludes with insights from research linking transformational leadership and diverse team performance.

2.3.1 Team Performance & Perceived Leader Support

Yukl and Lepsinger (1989) introduced the Managerial Practices Survey (MPS), a rigorous and comprehensive 14-category model of managerial practices. These 14 categories include:

1. Planning and organizing
2. Problem solving
3. Clarifying
4. Informing
5. Monitoring
6. Motivating & inspiring
7. Consulting
8. Recognizing
9. Supporting
10. Managing conflict and team building
11. Networking
12. Delegating
13. Developing and mentoring
14. Rewarding

.

Amabile et al. (2004) added further structure noting that these behaviors can have a *positive* polarity (i.e., they could be done right and/or have a positive outcome), a *negative* polarity (i.e., they could be done wrong and/or have a negative outcome), or a *neutral/other* polarity (i.e., they could have indeterminate outcomes). In their study of the day-to-day execution of these 14 behaviors, Amabile et al. (2004) found the following to be significant to perceived leader support:

- Monitoring
- Recognizing
- Supporting
- Rewarding
- Problem solving
- Motivating & inspiring

Each of these categories are discussed further below (note - while research contains an extensive debate on potential distinctions between management and leadership, the Amabile et al. (2004) study offers a convenient linkage that permits refraining from further discussion of the leadership-management debate in the scope of this thesis).

Monitoring. "Gathering information about work activities and external conditions affecting the work, checking on the progress and quality of the work, and evaluating the performance of individuals and the effectiveness of the organizational unit" (Yukl and Van Fleet, 1992, p.156). Amabile et al. (2004) found both the positive and negative version of monitoring behaviors to be significant to perceived leader support. That is, positive behaviors contributed to positive perceptions of leader support while negative versions contributed to negative perceptions. For example, positive behaviors included providing general guidance and positive constructive feedback as a result of regular contact with the team, whereas micromanaging or not understanding the work represented the category's negative version.

Recognizing. "Providing praise and recognition for effective performance, significant achievements, and special contributions" Yukl and Van Fleet (1992). Similar to the *monitoring* category, Amabile et al. (2004) found both positive and negative versions of this behavior to be significant to perceived leader support. Positive behaviors in this category included praising good performance in both public and private, whereas a negative behavior might include not recognizing performance at all or incorrectly recognizing the wrong person (Amabile et al., 2004).

Supporting. "Acting friendly and considerate, being patient and helpful, and showing sympathy and support when someone is upset or anxious, listening to complaints and problems, and looking out for someone's interests." (Yukl and Van Fleet, 1992). Amabile et al. (2004) found only positive versions of this category to be significant to perceived leader support. Examples of this type of behavior included leaders advocating on behalf of their team members' actions to higher-level leaders, finding ways to alleviate team member stress, addressing negative feelings on the team, and socializing with the team Amabile et al. (2004).

Rewarding. "Providing tangible rewards such as a pay increase or promotion for effective performance and demonstrated competence by a subordinate" (Yukl and Van Fleet, 1992). Similar to the *supporting* category, Amabile et al. (2004) found only positive versions of this category to play a significant role in perceived leader support.

Problem solving. "Identifying work-related problems, analyzing problems in a systematic but timely manner to determine causes and find solutions, and acting decisively to implement solutions and resolve crises" (Yukl and Van Fleet, 1992). Unlike the previous categories, Amabile et al. (2004) found only the negative versions of this category were significantly related to negative perceptions of leader support. That is, leader behaviors such as abstaining from getting further involved in solving problems or creating the problems themselves significantly contributed to negative perceptions of leader support (Amabile et al., 2004).

Motivating & Inspiring. "Using influence techniques that appeal to logic or emotion to generate enthusiasm for the work, commitment to task objectives, and compliance with requests for cooperation, resources, or assistance; also setting an example of proper behavior" (Yukl and Van Fleet, 1992). Similar to the *problem solving* category, Amabile et al. (2004) found only the negative version of behaviors to be significantly related to perceived leader support (or lack thereof, in this case). That is, leaders that were not authentic, for example, significantly contributed to a perceived lack of support among their teams Amabile et al. (2004).

In conjunction with the Amabile (1988) "componential theory of organizational creativity," the above study is important because it recognizes a tighter group of behaviors that leaders of diverse teams might focus on to enhance their team's creativity and ultimately improve their organizational success. Furthermore, as will be discussed further in the next section, some of these behaviors are similar to those associated with transformational leadership and may represent an even tighter group of behaviors to start with when embracing a transformational leadership style to improve team performance.

2.3.2 Leader Support & Transformational Leadership

Burns (1978) introduced the concept of transformational leaders as those that boost

the motivation, elevate the morale, and uplift the morals of their teams. They do so by helping to change their team members' values and mindsets such that they drive themselves to meet higher-level goals and needs beyond their own personal self interests (Amabile, 1983; Tierney et al., 1999; Jung, 2001). Additionally, transformational leadership embraces autonomy, empowerment, challenging work, and an innovative work environment, and it can therefore be an important mediator of job satisfaction (Bass (1999), creativity (Jung, 2001; Gumusluoglu and Ilsev, 2009), and team performance (Kearney and Gebert, 2009).

In particular, transformational leadership behaviors fall in one of the following categories:

- *Idealized Influence* - Behaviors associated with influencing dedication towards selfless ideals; they emphasize a collective dedication to the team's mission while also inspiring respect and admiration (also referred to as "charismatic role modeling") (Bass, 1999).

- *Inspirational Motivation* - Behaviors that communicate desirable visions and goals for the team, the paths to achieve them, and a confidence in the team members do so (Bass, 1999).

- *Intellectual Stimulation* - Behaviors associated with helping team members think about problems from different perspectives (Bass, 1985).

- *Individualized Consideration* - Behaviors associated with helping subordinates meet and exceed their own developmental needs, as well as to raise their standards of moral development (Bass, 1999).

Further, Tracey and Hinkin (1998) found that of the 14 Yukl and Van Fleet (1992) MPS categories, *clarifying*, *inspiring*, *supporting*, and *team building* represent the types of behaviors that most closely relate to the elements of transformational leadership. As introduced in Section 2.3.1, two of these behavior categories are significant to perceived leader support (i.e., *inspiring* and *supporting*), but only one of them in the positive form of the behavior (i.e., *supporting*) (Amabile et al., 2004; Tracey and Hinkin, 1998). In other words, of the behaviors associated with transformational leadership, it likely will be most important for leaders to perform the *supporting* behaviors correctly - while not performing the *inspiring* behaviors incorrectly - if they want to build the kind of support among their subordinates that leads to increased organizational success. This topic in relation to leaders of diverse teams is discussed further in the next section.

2.3.3 Transformational Leadership & Diverse Teams

Given the above relationships between team performance, perceived leadership support, and transformational leadership, and in light of the challenges associated with diverse teams discussed in Section 2.2, it stands to reason that transformational leadership may be an effective style for leaders of diverse teams to adopt.

For example, a transformational leader of a diverse team can use *motivate & inspire* and *idealized influence* behaviors to build a common and desirable membership-based social identity among their teams (van Knippenberg, 1999) while also highlighting the interdependence of the team members' responsibilities as they work together towards shared objectives (Shamir, 1990). Additionaly, *intellectual stimulation* behaviors are critical to diverse teams because they can help the team members be more open, accepting, and/or appreciative of the different perspectives they each offer (Shin and Zhou, 2007). Finally, with a more accepting environment, leaders that use *individualized consideration* behaviors among diverse teams can also encourage the members to be more expressive of their perspectives, including any concerns that might otherwise lead to disabling conflict among the team (Shin and Zhou, 2007).

In relation to this theory, scholars have found empirical cases for its validity. For example, Shin and Zhou (2007) studied team creativity among 44 South Korean firms' research and development teams that were heterogeneous from an educational specialty perspective, but highly homogeneous from an ethnicity and gender perspective (i.e., teams comprised primarily of South Korean males). Noting that heterogeneity in educational specialty enables the potential but does not guarantee greater team creativity (Milliken and Martins, 1996; Williams and O'Reilly III, 1998), they set out to determine if transformational leadership could reduce the negative effects of diversity while keeping the positives (Shin and Zhou, 2007). As their results show in Figure 2-4, transformational leadership behaviors can lead to greater team creativity among cognitively diverse teams.

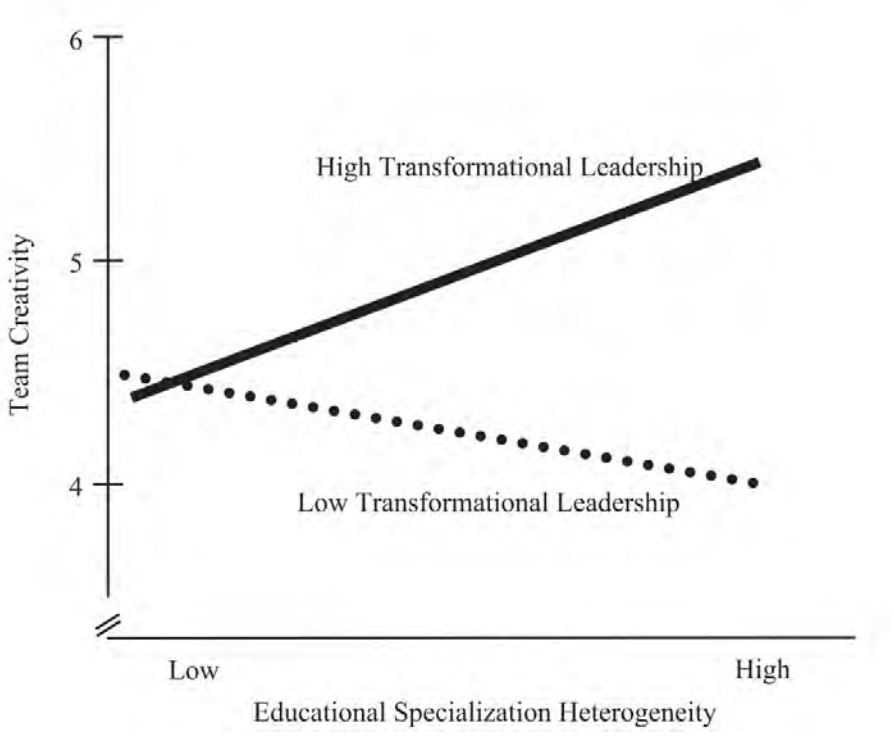

Figure 2-4: Transformational Leadership Impact on Team Creativity (Shin and Zhou, 2007)

Similarly, in addition to positively affecting cognitively-diverse teams, transformational leadership has also been shown to have a positive effect on the performance of identity-diverse teams as well. For example, Kearney and Gebert (2009) studied team performance (beyond creativity) among 62 research and development teams that were heterogeneous in terms of educational specialty, but in contrast to the Shin and Zhou (2007) study, were also heterogeneous in terms of age and nationality as well. Kearney and Gebert (2009) posit that transformational leadership leads to *collective team identification* and *elaboration of task-relevant information* as emergent states (Van Knippenberg et al., 2004). These emergent states, they argue, then counteract the deleterious effects of diversity found by focusing on social categorization and similarity/attraction theories (Williams and O'Reilly III, 1998) to ultimately yield greater team performance (Kearney and Gebert, 2009).

The results of the Kearney and Gebert (2009) study show that none of the diversity variables were **directly** correlated with improved team performance. However, the study did show that transformational leadership was positively correlated with the two proposed mediating emergent states (i.e. collective team identification and elaboration of task-relevant information), which indeed were positively linked to team performance (Kearney and Gebert, 2009). In other words, the diversity itself was not enough to make a significant impact on team performance without the transformational leadership as well. A sample of the Kearney and Gebert (2009) results is shown in Figure 2-5

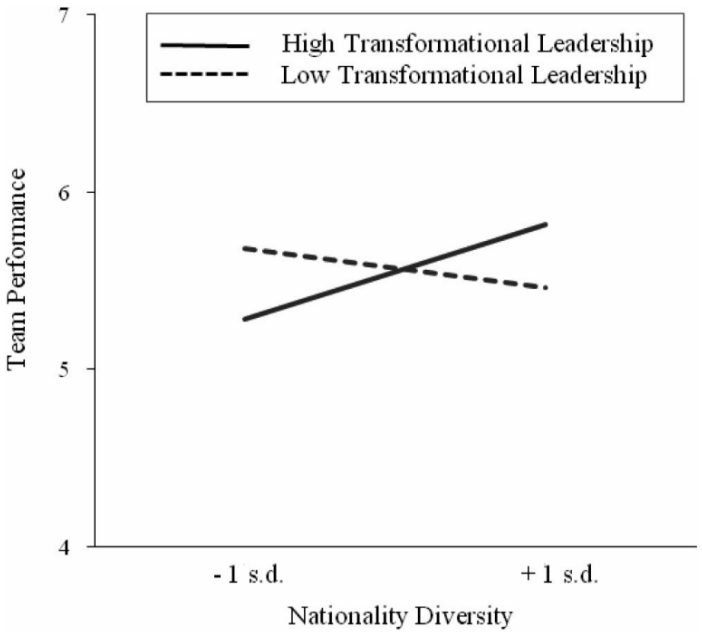

Figure 2-5: Transformational Leadership Impact on Team Performance (Kearney and Gebert, 2009)

As might be expected, however, not all empirical studies show elimination of the challenges associated with diverse teams. Kim (2017), for example, found that trans-

formational leadership did enhance team learning across 133 identity and cognitive-diverse teams from 5 companies, it was not enough to overcome the negative correlation between diversity and team learning. Additionally, Thayer et al. (2018), found that once a team transitions from its creative initiating stage to its implementation stage, it can be beneficial for leaders to also transition to a more transactional leadership style if creating is no longer a needed skill. Therefore, while this thesis is particularly interested in the impact of transformational leadership behaviors, it is likely not be the best leadership style for every situation.

2.4 Chapter Summary

This chapter first introduced the generally-accepted construct of teams as systems and concluded the topic with a recognition that today's teams are complex and dynamic. Therefore, it will be important for further research to account for this complexity and propensity for change. Then, by reviewing literature of diversity-based studies, we have a better understanding of the costs and challenges associated with diverse teams, as well as some insight into the mediating variables that can lead to positive team performance. Finally, by linking perceptions of leader support to team performance through the types of behaviors leaders perform, we can see how leaders of diverse teams might use the concepts of transformational leadership to achieve greater team success.

As I close this chapter in light of the research presented herein, I'd like to do so with a quote from Katherine Phillips that she penned in support of Scott Page's "Diversity Bonus" work:

> *"So, if there are so many clear and multifaceted benefits of identity and cognitive diversity in groups, why is there resistance or difficulty around increasing identity diversity in organizations? ... Diversity is difficult, and it is often hard to see the benefit in performance in organizations right away.... This can make it difficult to convince individuals that diversity is beneficial in comparison to the comfortable, more homogeneous environments that they have developed confidence in.... We should not stop chasing after the benefits of diversity because they are difficult to capture or because some are resistant. Diversity requires us all to work hard, and to embrace the change required in ourselves and in others around us." (Page, 2017, p.236)*

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Research Approach

> *"Whence all this passion towards conformity anyway? Diversity is the word." - Ralph Ellison in Invisible Man*

As introduced in Chapter 1, this thesis investigates the systems in which diverse teams operate in order to better understand actions that might strengthen retention and team performance. Unfortunately, as the research presented in Chapter 2 illustrates, the systems appear to be sufficiently complex such that simple and complete solutions may not exist. Therefore, in order to keep the scope of this thesis manageable, it limits its focus to supportive actions leaders of diverse teams can take to achieve the desired outcomes.

This chapter builds on the research presented in Chapter 2 and identifies a research question (Section 3.1) and potential hypothesis (Section 3.2) for further study. It then offers a potential empirical experiment design to address the hypothesis (Section 3.3. However, due to Corona Virus-19 limitations, an empirical experiment was not possible. Instead, I assessed available simulation approaches (Section 3.4) that might help address the research question in the place of an empirical study. The remainder of this thesis is devoted to the development and operation of one of these simulation approaches.

3.1 Research Question

There are a number of key points discovered in previous research that, when taken together, highlight potential opportunities for further exploration. The first of these points is the Apfelbaum et al. (2016) study indicating that an organization may be able to moderate the turnover of its underrepresented social groups via the approach it takes to diversity (i.e., *value in equality* vs *value in difference*). Since different social groups respond differently to the approaches, however, the question remains as to how leaders might close the gap such that they increase performance and retention of all social groups.

The next key point is the Reagans et al. (2004) study that provides insight into the unpredictability of diverse team performance due to the unknown strength of network pathways (i.e., *internal network density* and *external network range* involved. That

is, will the strength from a diverse team's larger external network range overcome the challenges that come with their typically weaker internal network density? Can leader behaviors strengthen internal network density such that the negative aspects of diversity are mitigated?

Next is the Kearney and Gebert (2009) study showing that transformational leadership can moderate a diverse team's performance by strengthening collective team identification as well as by elaborating on task-specific information. With the Amabile et al. (2004) research on the impact of perceived leader support and team performance due to various *daily* leader behaviors, coupled with the Tracey and Hinkin (1998) insights indicating *support* behaviors can be such significant positively-impacting behaviors for diverse teams, leaders might start here as they incorporate new actions into their day-to-day leadership style.

Finally, the Kozlowski and Chao (2018) and Mathieu et al. (2018) discussions regarding the dynamic nature of team systems, coupled with their collective advice to find ways to leverage today's technology for more frequent sociometric data from diverse teams, help frame a potential experiment to further understand diverse teams.

Therefore, with these insights in mind, this thesis attempts to address the following research question:

> ***Research Question*** **-** *How might transformational leadership behaviors associated with support affect a diverse team's internal network density and associated performance and organizational retention?*

The next section builds on the literature review offered in Chapter 2 and proposes a hypothesis to this research question.

3.2 Research Hypothesis

3.2.1 Combined Integrated Static Model

When integrated together, two previous models in particular appear to offer insight into a potential answer to the research question. The first is the network linkage model that Reagans et al. (2004) offer (see Figure 3-1), which as noted previously, suggests that team performance results from a combination of external network range and internal network density.

The other model is the Kearney and Gebert (2009) model showing how transformational leadership might moderate team performance (see Figure 3-2).

By integrating these two models, coupled with the research presented in Chapter 2, I offer the elaborated static model shown in Figure 3-3. While still a simplified abstraction, I offer this elaboration to suggest that transformational leadership *support* actions directed at building internal network density can build collective team identification, ultimately leading to improved team performance.

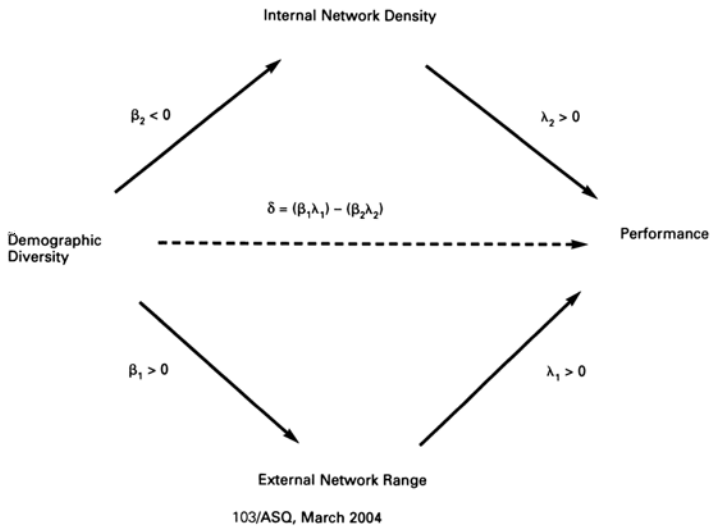

Figure 3-1: External Network Range and Internal Network Density Relationship with Performance (Reagans et al., 2004, p.103)

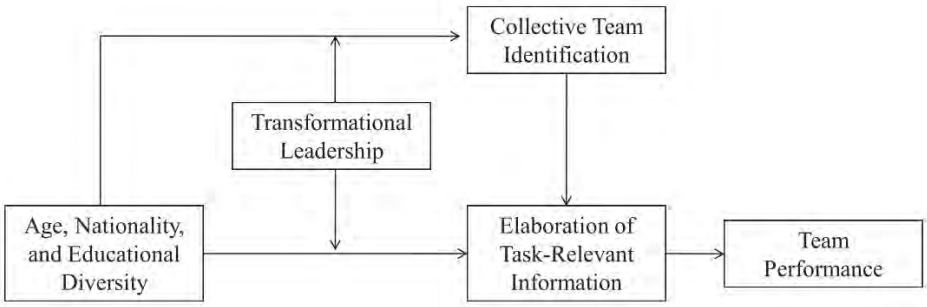

Figure 3-2: Relationship Among Transformational Leadership and Team Performance (Kearney and Gebert, 2009, p.78)

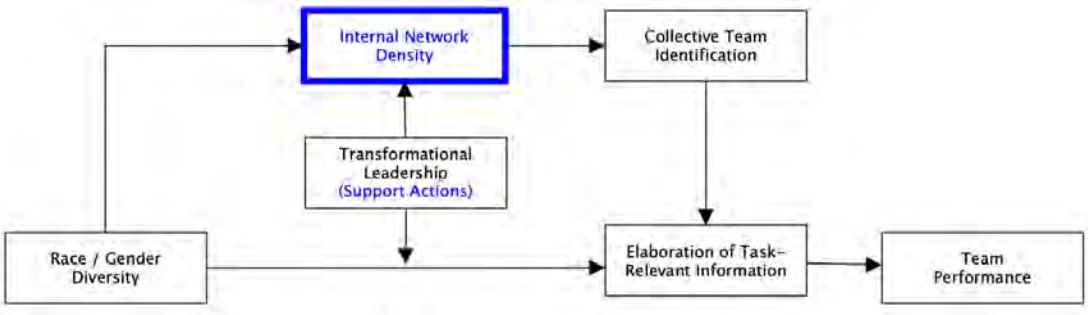

Figure 3-3: My Proposed Model Integration

3.2.2 Notional System Dynamics Representation

The integrated model shown in Figure 3-3 is helpful for understanding how team variables may be linked, but it does not depict the dynamics of their interactions. That is, it does not suggest how diversity concerns might be generated, it does not suggest when a leader should be performing leader support actions, and does it indicate any specifics on "team performance."

According to Reagans et al. (2004), one source of diversity-related concerns appears to be the reduced internal network density that results from adverse team member interactions. Further, as Amabile et al. (2004) notes, potential outcomes of these concerns include increased absences and higher turnover, which ultimately might lead to project delays associated with less available personnel. Therefore, especially for leaders not steeped in transformational leadership, I suggest that their initial support actions can be directed at those team members exhibiting the greatest symptoms of diversity-related concerns. For the purposes of this thesis, I limit discussion of these symptoms to a team's absences and turnovers. That is, subject to no other specific knowledge of a team member's diversity concerns, leaders may be able rely on observable outcomes like absences to signal a need to transition to supportive behaviors in order to prevent further negative results.

Additionally, if observable outcomes such as a team member's unusual absences can trigger such leadership action transitions, it stands to reason that they may also trigger a leader to pay more attention to others' supportive needs as well. In diverse teams, this might look like more daily interactions with team members in order to address their concerns more routinely.

Figure 3-4 depicts the dynamics of diverse team systems as I describe them above.

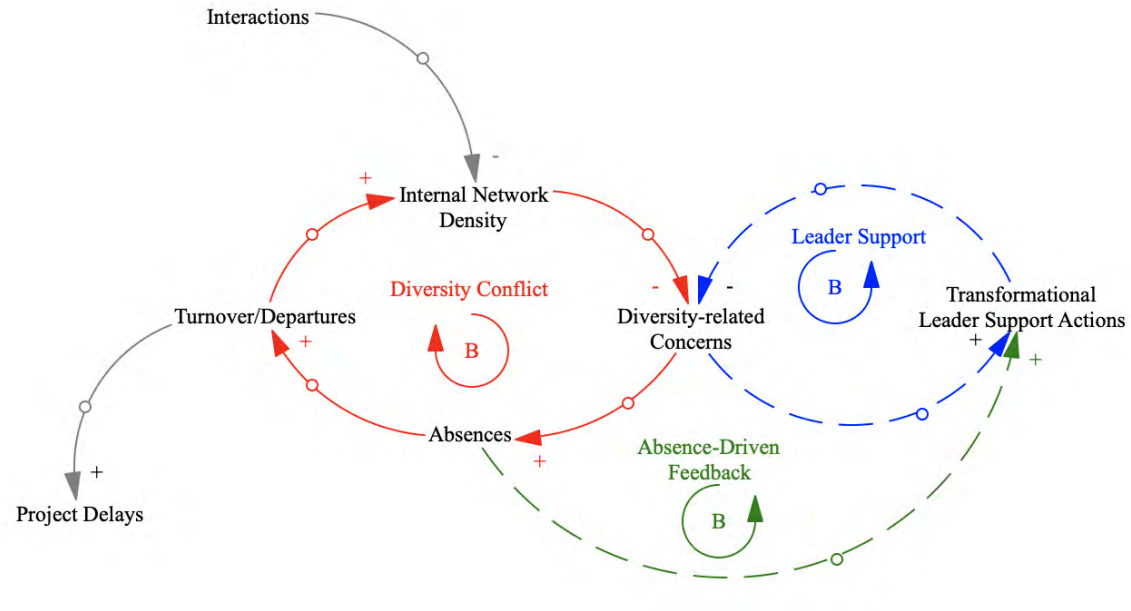

Figure 3-4: Proposed Systems Dynamics Model

3.2.3 Hypothesis

In light of the models presented above, I offer the following hypothesis in response to the research question posed in Section 3.1:

> ***Research Hypothesis -*** *a leader that executes supportive transformational leadership behaviors by spending time building support with members of his or her team will mitigate diversity concerns, thereby increasing retention and reducing project delays.*

3.3 Potential Research Experiment Design

This section provides an experiment design to test the above research hypothesis. Though this thesis did not perform an empirical data collection effort, the experiment design indeed corresponds to the simulation results presented in Chapter 5. I offer this section for consideration as a future research effort.

3.3.1 Objective

The primary objective of the experiment is to determine if transformational leadership *support* behaviors/actions can lead to decreased turnover and, by extension, decreased schedule delays of diverse team projects.

A secondary objective of the experiment is to collect dynamic team interaction data for correlation with diversity concern generation.

3.3.2 Domain

This experiment is designed for comparing three types of teams within one corporate setting. I propose to use the single corporate setting for all three teams in order to control for the diversity approach within which the teams are operating. All teams should be on the order of 10-20 participants, and should be composed of a low-represented social group (10%-20% representation), a moderately-represented group (30%-40%), and a highly-represented group (greater than 50%).

Each team should be clearly led by a single formal individual. In order to correlate transformational leadership support behaviors with turnover, it will be important that leaders can start the study without using them, learn more about them during the first part of the experiment, and then switch to using them at a point once predefined conditions are met. In order to control for transformational leadership behaviors, it will be important to establish a control team led by leaders that are not known for regularly practicing these behaviors. Additionally, though it is conceivable leaders already practiced in these behaviors can refrain from using them in the first part of the experiment, researchers should consider avoiding the use of these practiced leaders so as to not cause emotional stress to them or their team members during the less supportive phase of the experiment.

The projects each teams are working on should have time scales of approximately 90 days. Shorter projects may limit the measurable effectiveness of the supportive behaviors once leaders make their transitions. On the other hand, since time can also moderate conflict and other challenges of diverse teams (Pelled et al., 1999), projects of extensive length may confuse the effects of time vs. those of the transformational leadership behaviors.

Finally, multiple corporations should be approached to conduct this study in order to generalize the results. Though I am most interested in technical team dynamics, it is conceivable that non-technical firms with non-technical teams may be substituted as long as the above team size, leadership style, and task-relevant coordination stipulations can be met across all firms.

3.3.3 Data Collection

The experiment uses a combined data collection approach. One method uses sociometric badges to collect team member interaction data, while another method collects from the team members survey data to correlate with that of the sociometric badge data. This survey method consists of two surveys: 1) a daily one-question survey that give team members an opportunity to indicate when they generate diversity concerns (note - a corresponding survey will be available to leaders to generate when they address diversity concerns), and 2) a regular assessment of team member tie strengths to other team members (i.e., their internal network density). Finally, the researchers will need to work with the team leaders (or designated representatives) to track team member absences and turnover. These methods are explained further below.

Team Member Interaction. Team member interaction will be measured using sociometric badges (see Figure 3-5 for an example). These badges are wearable sensors that allow data collection of face-to-face interactions. Introduced by Alex "Sandy" Pentland of the MIT Media Lab, these badges will be used to anonymously collect the following data (with credit to Waber (2012) for consolidating these capabilities):

- Using 3-axis accelerometers to identify behaviors such as sitting or standing, running or walking, etc.
- Collecting "externals" of speech (i.e. not the actual words) such as amount, excitement, tone, patterns, etc.
- Measuring distance/proximity of individuals
- Using infrared to collect face-face interactions

Each team member will wear the badge for the entire length of the experiment.

Figure 3-5: Sociometric Badge (Waber, 2012)

Diversity Concerns. Additionally, another method of data collection is a quick, one-question daily survey to give team members an opportunity to respond to a single yes/no question regarding whether or not they experienced any diversity-related concerns over the day. Ideally, this data entry method will be available throughout the day such that the members can respond at a time closest to when they experience something that raises a concern.

Leader Support Behaviors. Recall from Yukl and Van Fleet (1992) that leader support behaviors include "listening to complaints and problems, and looking out for someone's interests." Therefore, leaders in this study will also need to respond to a single yes/no survey question when they implement these behaviors. Like members responding with diversity concerns as soon as they feel them, it will be most helpful if leaders also respond as soon after implementing the support behaviors as well.

Internal Network Density. While the sociometric badges will provide team member interaction data that can be used to determine changes in internal network density, it may prove important to track team member perceptions of these density changes as well. Therefore, the experiment should offer a regular (e.g., once per week) opportunity for team members to report changes in their tie strengths as well.

Absences/Turnover. Team leaders will need to take attendance - or ensure attendance is taken by someone on the team - on a daily basis and at each major team engagement in order to correlate diversity concerns with absences and turnover.

3.3.4 Execution

Pre-Experiment Setup. Consistent with guidelines from the Committee on the Use of Human Evaluation Subjects (COUHES), team members will need to provide informed consent to participate in this study. They should be informed that they will be participating in an organizational behavior study designed to correlate team member interaction with organizational approaches to diversity (i.e. *value in equality* vs. *value in difference*), and that in order to correlate results, their attendance will need to be taken at each major team engagement. Subject to COUHES approval, the

team members should NOT be specifically informed about the switch in leadership behavior aspect of the study in order to minimize any Hawthorne effects from being introduced.

During this phase, leaders of the different teams being considered for the study should be informed of the intent to study the impacts of using transformational leadership support behaviors on team performance and persistence. They should also be given the Multifactor Leadership Questionnaire (MLQ) developed by Bass and Avolio (1990) to determine if they will be good candidates for the study.[1] Finally, they also should be given the opportunity to opt their teams out of the study if they have concerns with being able to distinctly lead them under two different support-related styles.

Once leaders are selected, the researchers should establish three types of teams. The first is a control team (i.e., "control team") which will not be given any direction on how to lead their teams. The next is a team where the leader specifically tracks absences from work and/or team engagements (i.e., "observable team"). Finally, the last type of team will allow researchers to give the team leader insights from the sociometric badge data such that the leader can make informed decisions on when supportive actions are needed (i.e., "badge awareness team").

Once teams are selected, team members should be polled to first determine the strength of their individual ties to one another as performed in Reagans et al. (2004). The respective team internal network densities can then be calculated with the following equation (Reagans et al., 2004):

$$ND_k = \sum_{i=1}^{N_k} \sum_{j=1}^{N_k} \frac{z_{ij}}{max(z_{i_q})} / N_k(N_k - 1) \tag{3.1}$$

where

$$\begin{gathered} ND_k = \text{Average internal network density} \\ i = \text{each respective team member} \\ j = \text{every other team member} \\ z_{ij} = \text{tie strength between members } i \text{ \& } j \\ z_{iq} = \text{team member } i\text{'s maximum tie strength on the team} \\ N_k = \text{number of members on the team} \end{gathered}$$

Finally, each team member will be given a sociometric badge and will be informed of how to submit its data for study collection. They will also be told how to register their daily one-question diversity concern study, as well as when and how updates to internal network density will be collected. The members will then be released to start their work.

Experiment. Aside from transferring sociometric badge data, the daily surveys, and the regular internal network density updates, team members will have no other responsibilities in this study. This is not true for the leaders, however.

[1] see https://www.mindgarden.com/16-multifactor-leadership-questionnaire for more on the MLQ

"Observable Team" Leader Actions. Leaders of the observable team will establish an absence threshold with the researchers for when to make the switch to supportive behaviors. If that threshold is reached, they will gather the entire team together to inform the members that a study threshold was reached, and they will use this trigger to collectively understand what diversity (and other, if appropriate) concerns are impacting the team. This may be a delicate situation, so the leader may need to use his or her own discretion in collecting this information. The leader will then transition to providing more daily support to the team to ensure the concerns are addressed.

"Badge Awareness Team" Leader Actions. As correlation of sociometric badge and one-question survey data begin to reveal diversity concerns, researchers will inform the team leader that support actions are required (not by whom, however, in order to protect privacy). In contrast to the intervention-style response of the "observable team" leaders, however, the "badge awareness team" leaders will instead transition to more daily support actions with team members that they perceive/learn to need more specific support.

Post-experiment Data Correlation Though researchers should be correlating sociometric badge data with diversity concern data throughout the experiment, conclusion of the experiment will also allow the absence/turnover data and the internal network density data to be analyzed as well.

3.4 Simulation Approach

Finally, as noted in the chapter introduction, this thesis leverages the capability of models to substitute for empirical data collection. With a number of different types of models available, however, it was important to consider what type of model to use. For example, system dynamics models and agent-based models (ABMs) are important pieces of the System Design & Management curriculum. Would one be better than the other? Would other model types be even more appropriate? This section briefly summarizes the available modeling methods, and then closes with a discussion on the selected modeling approach.

3.4.1 Available Modeling Methods

The National Research Council of the National Academies consolidated the various types of modeling approaches, coupled with their respective strengths and weaknesses, as part of a study sponsored by the United States Air Force (NRC, 2008). According to their survey, the types of modeling approaches include, each of which are discussed further below:

- Verbal conceptual and cultural models
- Macro-level formal models

- Micro-level formal models
- Meso-level formal models

Verbal Conceptual and Cultural Models. Verbal conceptual models use natural language to describe the relationships among the different entities, processes, and variables of the phenomena being modeled. They can eloquently express ideas (e.g. "brain as a computer"), but that strength is also a weakness in that different people can interpret them in different ways (NRC, 2008, 102). For example, using the "brain as a computer" verbal model, one might interpret this in terms of memory, processing speed, the tendency to get bogged down with unimportant tasks, or even the need to be reset every now and then. Similarly, verbal models can also be difficult to validate, and their use may simply rely on surface-level validity (NRC, 2008, p.103).

Verbal cultural models use natural language to describe an entity's culture, such as a region's language and history (NRC, 2008, p.104). Less concerned with how a culture came to be (which is often more a topic of all other types of models), the four different types of verbal cultural models include: "cultural inventory models, dominant trait models, semantic models, and cultural domain models" (NRC, 2008, p.105).

While these types of models may elicit unique aspects of diverse team leadership, they are not suitable for demonstrating the dynamic phenomena of the research question. They are therefore not appropriate for this particular thesis effort.

Macro-Level Formal Models. There are two types of macro-level formal models: system dynamics models and organizational models (NRC, 2008, p.122). System dynamics models, like the one shown in Figure 3-4, were introduced by Professor Jay W. Forrestor of MIT in the 1950s to model dynamic states and interactions of different entities of a system such that one entity's outputs become another entity's (or multiple entities') inputs (NRC, 2008, p.129). One particular use of system dynamics is to model how quantities of certain system entities (i.e. "stocks") accumulate or deplete (i.e., "flow") as different aspects of the system affect their rates of change.

For example, a system dynamics model could have been generated to address the research question posed in this thesis. Figure 3-4 represents a good start, though it would need to be adjusted to show team members as the stocks (classically shown via a box around it) and departing team members as a flow (classically shown with a valve/hourglass icon). The various system variables would then need to be appropriately adjusted to reflect their respective interactions with surrounding variables. This model would then represent a good macro view of the system and indeed, due to the single team implementation of the model built for this thesis, likely would produce similar simulation results as will be shown in Chapter 5. However, in order to posture this work for further meso-level analysis that will be discussed later, a quantitative system dynamics model was not pursued at this time.

Another type of macro-level modeling is the organizational model (NRC, 2008, p.135). In these efforts, organizations are intricately modeled to reflect who does what

tasks, when they do them, and where they do them in order to address organizational-level questions (NRC, 2008, p.135). For example, one might model how decisions are made in order to address the potential need to centralize or decentralize the organization's decision-making progress.

ABMs can also be organizational models if the agents are coded with the rules and communication lines of the organization. In the case of this thesis, organizational rules are not necessarily in question, so this type of model was also ruled out for further action.

Micro-Level Formal Models. According to NRC (2008), different types of micro-level models are used to study behavior of individuals. These models include cognitive architectures, cognitive-affective models, expert systems, decision theory, and game theory (NRC, 2008, p.149). Due to the nature of these models being designed for individual-level study, they were not considered to address the team-based nature of this present effort's research question.

Cognitive architectures broadly simulate individual human information processing to assess cognition aspects such as memory and learning (NRC, 2008, p.150) as they affect individual behavior. Their most significant limitation, however, is that they can be extremely difficult to verify and validate due to the extensive knowledge engineering they require (NRC, 2008, p.164).

Cognitive-affective architectures also simulate human information processing but include aspects of how emotions contribute to the processing as well (NRC, 2008, p.168). Though not selected for this thesis effort, it is conceivable that due to the emotions involved with diversity-related concerns, a cognitive-affective architecture could be built to study micro-level impacts of transformational leadership behaviors on human processing.

Expert systems are software programs with significant amounts of relevant knowledge in their knowledge bases that are used in conjunction with with a working memory and an inference engine to simulate specialized tasks like medical diagnoses (NRC, 2008, p.185). They are useful in making decisions, training, and preserving knowledge in particular areas (NRC, 2008, p.187).

Decision theory models try to maximize a *payoff function* as participants choose from a possible actions (NRC, 2008, p.195). The payoff is then a function of the state of the system applied to the participant's decision (NRC, 2008, p.196). Game theory models are very similar, with the notable exception that an opponent also gets to make a decision from a possible set of actions (NRC, 2008, p.199).

Meso-Level Formal Models. Meso-level models are those that study phenomena between the micro and macro levels, such as individual contributions to overall group performance (NRC, 2008, p.215). Models at this level include voting and social decision models, social network models, and ABMs (NRC, 2008, p.215). Note that because this effort uses an ABM, I have intentionally broken it out into its own separate section below.

Voting and social decision models leverage voting theory to model how individual

votes (note - not limited to formal electoral voting processes) will drive overall group results (NRC, 2008, p.216). Because voting is not germane to the research question, such models were not pursued further.

Social network models conceive individuals in groups as a set of nodes that have ties to others in the group to form a network (NRC, 2008, p.222). As internal network density is a specific aspect of this research effort, this type of model was a conceivable option. However, it was deemed not the best option due to this type of model's tendency to focus on the actual network itself rather than the characteristics of the nodes (NRC, 2008, p.223). As race and gender are important characteristics considered in this effort, modeling the characteristics of the nodes was more important at this time. However, it is conceivable that further effort can be applied in this area.

3.4.2 Selected Approach: Meso-level Agent-Based Model.

Agent-based models simulate macro-level emergent outcomes as a result of micro-level agent behaviors (NRC, 2008, p.236). Specifically, they can simulate heterogeneous individuals, each with their own internal states that can vary over time as they produce macro-level results (NRC, 2008, p.237). This section briefly summarizes the benefits of ABMs and their most significant limitation.

Benefits of ABM-related Research. ABMs are suited for the following types of research goals (NRC, 2008, p.239):

- Empirical description - helping to answer why macro-level structures have emerged.
- Empirical prediction - helping to identify potential future consequences that might emerge from programmed scenarios.
- Normative analysis - helping to identify what rules of social systems might lead to desirable outcomes.
- Behavioral understanding - helping to address how different individual behaviors interact with others to create macro-level impacts.

Major Limitation. The most significant limitation of ABMs is the difficulty in getting the level of realism correct (NRC, 2008, p.247). Some models are oversimplified and thus should not be used for decision making, while others may involve so many rules that the model becomes overfitting and can be made to reflect any possible outcome (NRC, 2008, p.248). This thesis attempts to achieve a balanced level of realism from which to derive further insights.

Decision Rationale. The relationship between organizational outcomes and individual diversity concerns studied under this effort's research question is suited to ABM strengths. Additionally, though this thesis is limited to the outcomes associated with leader support-related behaviors, it is conceivable that other behaviors and associated impacts can be incorporated in future updates. Therefore, in conjunction with

the draft experiment design discussed in Section 3.3, the ABM is a good candidate for further exploration.

3.5 Chapter Summary

After reviewing the past research in Chapter 2, this chapter poses a new question intended to bridge the more static nature of past research to a new era of dynamic research. The chapter then posits a hypothesis and offers an experimental design to address that hypothesis. Without an option to conduct such an experiment during the Corona Virus-19 pandemic, the remainder of the chapter discusses potential simulation options to address the research question. Finally, the chapter closes with rationale for building and using an agent-based model to simulate the experiment discussed earlier.

The next chapter presents the relevant details of the ABM built to simulate the experiment.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Agent-Based Simulation

> *"The behaviors that produce diversity bonuses must emerge from the bottom up, organically."* (Page, 2017, p.5)

The cornerstone of this thesis effort is an agent-based model built to simulate the experiment design proposed in Section 3.3. This chapter provides the relevant details of the model, to include the assumptions necessary to keep the work moving forward. It introduces the model with a general overview followed by a detailed description of the major class files. Additionally, the entire code is presented in Appendix A for future development consideration.

4.1 Assumptions

For transparency purposes, this section consolidates the assumptions I made in the creation of this model.

- Leader demographics do not affect team dynamics. This assumption is likely not realistic; however, the topic is outside the scope of this thesis. Therefore, it is not modeled in this iteration.
- Individual agent network ties are symmetric.
- Individual agent network ties initially range from 0.25 to 0.95.
- Agents with similar race AND gender get a 0.25 bias in tie strength to account for similarity/homophily.
- Agents with similar race OR gender get a 0.125 bias in tie strength to account for similarity/homophily.
- Agents will need to work with dissimilar agents in order to accomplish coordination-required tasks. This simulates agents having different skill sets required for task completion.
- Agents will register diversity concerns either from task interactions or from operating in a diversity approach prone for concerns from their demographics.
- There exists a threshold at which accumulation of diversity concerns will trigger agents to go inactive for the period. This simulation assumes that threshold is 10% of the project span.

- There exists a threshold at which accumulation of agent absences will trigger agents to go inactive for the remainder of the simulation. This simulation assumes that threshold is also 10% of the project span.
- Leaders can address lingering diversity concerns even after an indefinite time has passed. Though it is likely that some concerns will go to a permanent "ignored" status, the scope of this thesis did not include such dynamics. Therefore, these dynamics will be left for future model iterations.
- A full-day leader "intervention" is sufficient to address the diversity concerns of a single agent that is close to reaching the absence threshold. This intervention simulates the leader pulling the team away from work (e.g., an "offsite" meeting) to begin hearing and addressing concerns across the team.
- The model currently assumes that leaders effectively mitigate diversity concerns whenever they attempt to. This includes both in full team interventions, as well as in daily support actions. This is an unrealistic assumption; however, the scope of the literature review did not address leader effectiveness for this thesis. It will therefore be left for future work.

4.2 Model Introduction

Figure 4-1 introduces the object classes built for the ABM. The basic functionality of each of these files is provided below, and the next section will illustrate how the classes interact.

The class files are organized into three main categories. The first is the user interface controllers, of which there are two types: one to control a single simulation run, and one to control a Monte Carlo simulation that executes a specified number of simulation runs. The second category is the agent environment, and the third category is associated with individual agents.

The actual files in these categories are discussed further below.

- User Interface Controller
  - Inputs (*UserInterfaceInputView.java* and *UserInterfaceInputViewMonteCarlo.java*): Allows a user to specify certain simulation parameters, respectively.
  - Results (*UserInterfaceResultsView.java* and *UserInterfaceResultsViewMonteCarlo.java*): Displays the results of the simulation, respectively.
  - User Interface (*UserInterface.java* and *UserInterfaceMonteCarlo.java*): Combines the input and results classes to display a single user interface. Sends input parameters to the Model Runners, respectively.
- Environment
  - Model Runner (*DiversityModelRunner.java* and *DiversityMonteCarloSim.java*): collects parameters from the user interface and establishes a session for the

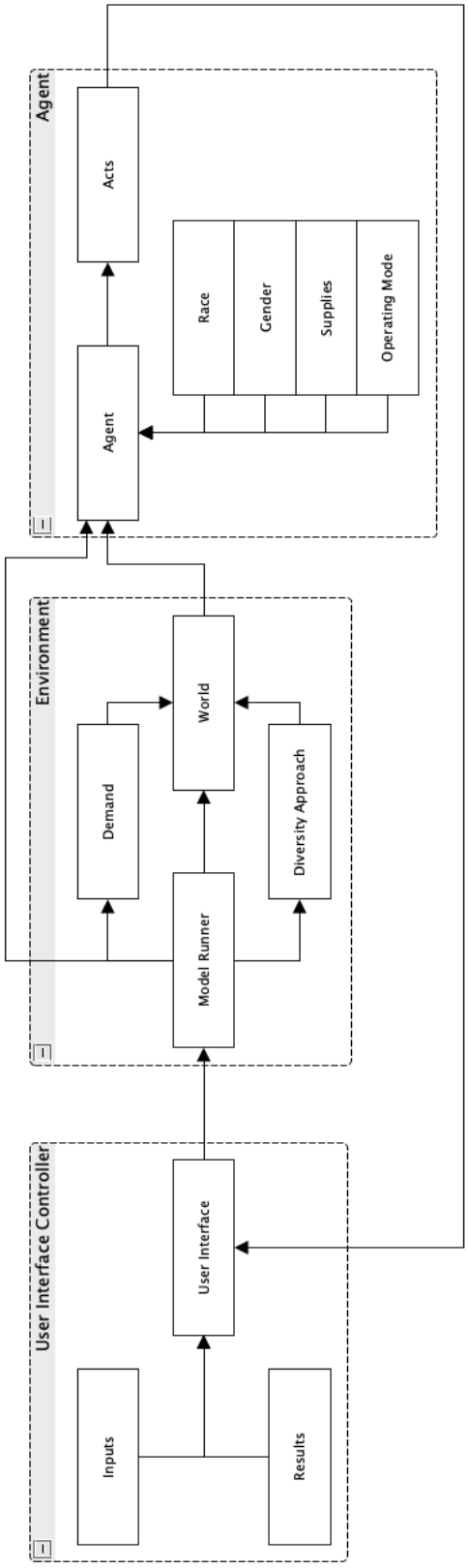

Figure 4-1: Model Class Map

simulation, respectively. Creates agents, initial demands, and a "World" for the agents to operate in. Passes parameters such as the user-defined diversity approach to the World.

  - World (*World.java*): the environment the agents operate in.
  - Diversity Approach (*DiversityApproach.java*): enumerates the different types of diversity approaches available to simulate. This initial model only includes *value in equality* and *value in difference*.
  - Demand (*DemandFX.java*): a class to establish demands that agents will work to satisfy.
  - Simulation Result (*SimResult.java*) A collection of simulation results from each run (single result for single simulation, and a collection of all results for the Monte Carlo simulation).

- Agent
  - Agent (*Agent.java*): holds the state of the agents and defines the rules in which it must operate. Performs "acts" to satisfy demands.
  - Race (*Race.java*): an enumeration of available races for simulation. This initial model only includes "high representation" and "low representation" consistent with the work of Apfelbaum et al. (2016).
  - Gender (*Gender.java*): an enumeration of available genders for simulation. Like its Race counterpart, this initial model only includes "high representation" and "low representation."
  - Supplies (*SupplyFX.java*): a class file of available resources for agents to leverage when satisfying demands.
  - Operating Mode (*AgentMode.java*): an enumeration of available modes the agent can be in. Modes of this initial model include INACTIVE, OBSERVING, SELECTING, ACTING, SUPPORTING, FINISHING, RECOVERING, WAITING, COMPLETING, PAUSED, TIMELIMITED, and DEPARTED.
  - Act (*ActFX.java*): a class file to define what types of acts the agents can use to match their supplies with the demands they are trying to meet. Results of acts are collected and displayed in the User Interface.

With this introduction established, the next section provides further details on the key objects built in this model.

4.3 Detailed Model Functionality

4.3.1 User Interface (Single Simulation & Monte Carlo Simulation

Summary. The simulation uses user interfaces to not only simplify input adjustments, but also to more easily visualize the dynamics of the simulations over time.

The user interfaces, each respectively shown in Figure 4-2 and Figure 4-3, are each divided in two sections. On the left side, a user can set the simulation parameters, and the right side is used to show the respective simulation results. The entire code for each of these interfaces, including .fxml files for the graphics themselves, is presented in Appendix A.1.

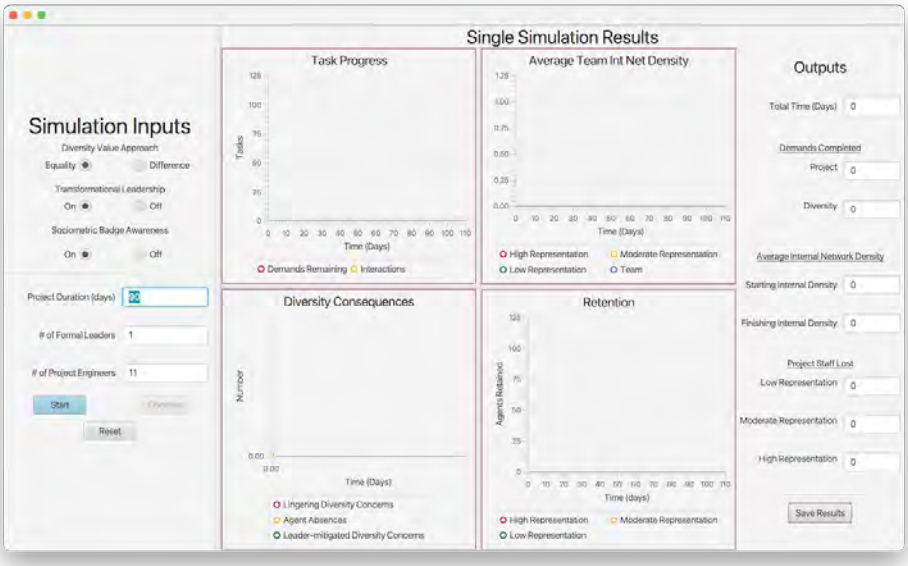

Figure 4-2: Single Simulation User Interface

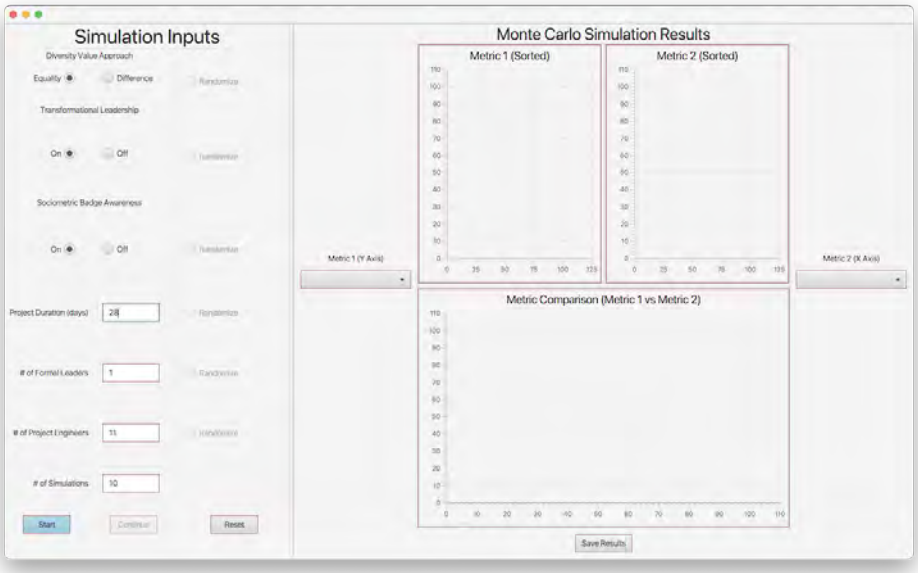

Figure 4-3: Monte Carlo Simulation User Interface

Each side of the user interface is detailed further below.

Simulation Inputs. The current iteration of the model includes six user-defined parameters. The first is the **Diversity Value Approach**, which can be set to either *Value in equality* or *Value in difference*. Per the Apfelbaum et al. (2016) research, this setting establishes the environment shown to cause different concerns depending on agent demographics.

The **Transformational Leadership** setting can be set to *On* or *Off*, which allows the user to turn a leader's supportive behaviors on or off for the particular simulation.

Additionally, the **Sociometric Badge Awareness** setting can also be set to *On* or *Off*. In accordance with the experiment defined in Section 3.3, this setting is included to simulate daily leader support actions as a result of researcher-provided insights from the sociometric badges.

Next, the **Project Duration (Days)** setting defines the amount of work required to complete the project for comparison with the **Total Time (Days)** result discussed below. This setting also establishes the initial number of demands (or tasks) for the agents to complete. Engineers are expected to complete eight 1-hour tasks design tasks per day on the project. Similarly, Leaders are expected to complete six 1-hour work tasks, leaving two time blocks per day for leadership-oriented tasks.

The # **of Formal Leaders** setting creates the specified number of leaders, with randomly-established race and gender demographics (note - leader demographics are not included in this iteration of the simulation, thus representing an important limitation of the current model). Additionally, the current iteration of the model only includes functionality to model INTRA-team interaction, but I envision building in further capability to model INTER-team interaction among other teams as well. I also anticipate giving engineers "LEADERSHIP" skill sets in order to show the impacts of informal leaders on the team as well. Until this functionality is built in, however, changing the number of formal leaders is not expected to provide realistic results at this time.

The last setting in the Single Simulation User Interface is the # **of Formal Engineers**. The code is designed such that setting this value at 11 establishes a test case with agents meeting the following demographics:

- 6 engineers in both high-representation racial and gender social groups.
- 3 engineers in a high-representation racial social group and low-representation gender group.
- 1 engineer in both low-representation racial and gender social groups
- 1 engineer in a low-representation racial social group and a high-representation gender group.

Setting the number of engineers to any other value will randomize both the race and gender of the agents.

Finally, the Monte Carlo Simulation User Interface also includes a text field to specify how many runs to execute. There are no specified limits to this value. Additionally, while the code does not currently allow a user to randomize each variable, this user interface includes placeholders for enabling this functionality in the future.

Single Simulation Results. The right side of the user interface displays four graphs as well as a number of text fields for specific results.

The graph in the upper left is the **Task Progress** graph. Here, two data series are tracked. The first shows the burndown of initial tasks, while the second illustrates when coordinations occur.

To the top right is the **Average Team Int Net Density** graph that tracks the team's average internal network density, as well as the average internal network density of each social group. Each data series represents the dynamic average of each respective set of agent average tie strengths.

The graph in the bottom left presents the simulation's **Diversity Consequences**. Three data series are presented here. The first data series captures the diversity concerns that are still lingering after the leader support actions (note - if transformational leadership is off, this data series presents all of the generated diversity concerns), the second is the resulting absences from those diversity concerns, and the third captures the leader's actions to mitigate those concerns if transformational leadership is turned on.

Next, the graph to the bottom right shows the retention of agents throughout the simulation. The data are collected in terms of low-, moderate-, and high-representation social groups for comparison with the Apfelbaum et al. (2016) results.

Beyond the charts to the far right of the user interface are specific text fields. The first is the **Total Time (Days)** total actual time to complete the project.

Next is a set of blocks to show how many total **Project** demands/tasks were completed, as well as how many **Diversity** concerns the leader mitigated (not necessarily how many were generated).

Below these are the **Starting Internal Density** and the **Finishing Internal Density**, which are the start and finish average team values, respectively.

Finally, the last block displays how many agents in each demographic social group decided to leave the team.

Monte Carlo Simulation Results. The Monte Carlo simulation uses three charts to display two metrics. The two charts on the top of the interface display a user-selected metric sorted from its least to greatest value in order to understand how many times a result occurred during the simulation (though not a histogram). The first metric is selected via a drop-down box on the left side of the charts, and the second is selected via a drop-down box on the right side.

In the bottom chart, the user-selected metrics are plotted in comparison with each other in order to understand their respective correlation. Specifically, Metric 1 is plotted on the Y-axis, and Metric 2 is plotted along the X-axis.

The user-selected metric options include:

1. Interactions
2. Final Internal Network Density (IND)
3. Diversity Concerns
4. Agent Absences
5. Leader Interventions
6. Leader Daily Supports
7. Agents Lost (Low)

8. Agents Lost (Moderate)
9. Agents Lost (High)
10. Project Duration

Finally, an option to save the results in Microsoft Excel® format is provided under the bottom chart.

4.3.2 Diversity Model Runner & Diversity Monte Carlo Simulation

Summary. In order to ultimately allow multiple agent environments (see "World" in Section 4.3.3 below), the model contains an abstraction one level higher than a single agent environment to run the execute the overall simulation. The Diversity Model Runner class ("DiversityModelRunner.java", full code presented in Appendix A.2.1) initializes the variables established in the user interface. It also creates the initial set of global demands, the agents to accomplish them, and the world for the agents to operate in. Finally, it offers a "reset" method to clear variables and rerun simulations directly from the user interface.

Similarly, the Diversity Monte Carlo Simulation ("DiversityMonteCarloSim.java", full code presented in Appendix A.2.2) initializes the same variables for this multiple-run version of the simulation. Like its counterpart, it creates an initial set of demands, the agents to accomplish them, and the world for the agents to operate in. It also offers a "Reset" button for conveniently resetting and re-executing the simulation. Unlike its counterpart, however, this class also collects SimResult data for display in the Monte Carlo user interface.

The above-mentioned differences represent only significant differences between the Single Simulation and the Monte Carlo Simulations. The remainder of the code discussed below, except where otherwise noted, applies to both of these simulations.

4.3.3 World

Summary. The current model enables simulation of a single organization and internal team. The World class ("World.java", full code presented in Appendix A.2.3) simulates the organization and establishes the environment the agents operate in. During Monte Carlo simulations, a new randomized World class is established for each individual run.

The "World" class also establishes simulation aspects that would not reasonably be established by the agents. For example, it simulates the formation of the team and establishes an initial tie strength between each of the agents. Similarly, it also calculates and tracks the team's average internal network density throughout the simulation. It is also the class that tracks overall project status, thus allowing the agents a place to observe remaining work.

Finally, at the end of each time step (currently set to 1 day), the "World" class collects the agents' results for display in the single simulation user interface.

The class' primary methods are detailed further below.

Execution Control Methods. The "World" class contains two execution control methods:

1. start()
2. continueSim()

The **start()** method is responsible for agent initialization as discussed in the summary. It also captures status and applicable results for the user interface and the output screen. It is only performed once per simulation.

The **continueSim()** method, similar to the **start()** method, controls the output to the screen and user interface. It also controls the simulation execution past the first time step. It checks to see if there are any remaining demands, updates the team's internal network density from the previous time step for tracking purposes, and sets the agents working again. At the end of the period, it re-initiates any agents that have taken the day off, as well as collects all the departed agents to ensure they can no longer contribute to task execution.

Output Control Methods. The "World" class contains four output control methods:

1. printStartLoop()
2. printSimResults()
3. printDemandStatus()
4. updateUI()

While the class has a number of convenience methods associated with displaying simulation results, the most important of these methods is the **updateUI()** method. This is where each time step's results are added to respective data series so the user interface can collect and display them. This method also controls a collection of screen outputs, as well. The remaining methods are included in this final submission due to their usefulness in testing code updates; however, note that their outputs are commented off in the code provided in the Appendix.

Agent Convenience Methods. The "World" class contains five convenience methods to help with tracking the agents:

1. refreshAgents()
2. retireAgents()
3. getAgentConnection()
4. refreshTeamIntNetworkDensity()
5. addAgent()

The **refreshAgents()** method is a convenience method to ensure agents in *RECOVERING* mode (i.e., taking the time period off) are brought back to work in the next time period.

The **retireAgents()** method is a convenience method to collect agents in *DEPARTED* mode and removes them from the list of agents as they return to each work time period.

The **getAgentConnection()** method is used to establish initial tie strength between the agents. It randomly assigns a value between 0.25 and 0.95 for each agent

pair. Further, as discussed in the Assumptions (Section 4.1) biases the strength an additional 0.25 if both race and gender match, or just 0.125 if either race or gender match.

The **refreshTeamIntNetworkDensity()** method uses Equation 3.1 to calculate the current average team internal network density for user interface tracking purposes.

Counting Convenience Methods. The model contains several convenience methods to help with counting results for display in the user interface. They are listed here for completeness, but will not be discussed further due to their relative insignificance.

1. checkAnyDemandList()
2. countStateDemands()
3. countTypeDemandsRemaining()
4. countDiversityConsequences()
5. countLeadershipTasks()
6. countCoordRequests()
7. countAgents()
8. countAllAbsences()
9. getDiversityTrend()

4.3.4 Agent

Summary. The Agent class ("Agent.java", full code presented in Appendix A.3.1) simulates individual team members, and it controls how agents operate. It governs how agents work together on coordination-required demands, and it sets the rules under which the agents generate signals indicating diversity concerns. Similarly, it also controls whether or not they decide to "stay home" for the period, or if they will depart the simulation altogether. Finally, the class also governs the agent response to leader-initiated (leader is a type of agent) concern mitigation efforts.

Additionally, there are two types of agents in the existing simulation. The first is the leader, and the other is the Engineer. Both have different supplies that the "Agent" class stipulates in order to address different types of demands. Because of this distinction, the class has a number of rules that are only for Leaders to follow (e.g. checking to see if the team is struggling and needs support).

Simulation Execution Control Methods. The "Agent" class contains two execution control methods:

1. start()
2. continueSim()

Similar to the "World" class, the "Agent" class has corresponding **start()** and **continueSim()** methods. The **start()** method initializes the agent with an initial time stamp, calculates its average internal network density, and then starts the agent on its execution (**loop()**). When it is done executing its period's work, the **start()** method calls the **generateRandomDiversityConcern()** method (discussed further below). This simulates team members reporting once per day that they experienced some sort of diversity-related negative experience that day. The **continueSim()** method works in a similar manner, with the one exception that it also resets that agent's supplies for each period.

Agent Control Loop. The "Agent" class contains a control loop method that uses the agent's current mode to direct how the agent flows through the period. For example, a typical period might occur along these lines:

1. Agent starts in *INACTIVE* mode.

2. Agent switches to *OBSERVING* mode and executes **Observe()** method (discussed below). Agent leaves this method with a task list for the time block.

3. Agent switches to *SELECTING* mode and executes **Select()** method to attempt to select a demand to satisfy. Agent typically leaves this method with a demand matched to its supply.

4. Agent switches to *ACTING* mode and executes **Act()** method to act on the demand and update its supplies. Agent leaves this method with the need to finish the task and capture its results.

5. Agent switches to *FINISHING* mode and closes out the task. Agent leaves this method with a shorter task list and goes returns to *SELECTING* mode to get its next task.

In non-typical periods, the **loop()** method also allows for the agent to transfer its mode to *RECOVERING* and *DEPARTED*, which in turn give the day off or retire the agent, respectively. Finally, it also allows Leader agents to transfer to *SUPPORTING* mode in order to support the Engineer agents.

Methods by Agent Mode. As introduced in the Agent Control Loop section above, the "Agent" class contains nine methods executed according to the agent's current mode:

1. Wait()
2. Observe()
3. Select()
4. Act()
5. Finish()
6. StayHome()
7. Depart()
8. Support()
9. Complete()

The **Observe()** method is where the agent observes and orients to its environment. Here, agents first assess their unaddressed diversity concerns and absences, and if both are above the respectively established thresholds, the agents switch to *DEPARTED* mode and can no longer work on the project. Agents that do not depart then check to see if they need a day off. Assuming they come to work, this method loads up their task list for the period.

Additionally, the **Observe()** method is also where Leader agents will check to see if their teams need support (assuming "Transformational Leadership" is turned on in the user interface). If so, here is where the agent switches to *SUPPORTING* mode to execute the **Support()** method.

The **Select()**, **Act()**, and **Finish()** modes function as described in the Agent Control Loop section above and do not need further discussion here.

The **StayHome()** method pauses the agent and fast-forwards its time stamp one full period when results from the **generateRandomDayOff()** method (described in the Diversity Consequences Methods section below) are "true".

Similarly, the **Depart()** method leverages the results of **generateAgentDeparture()** method. In this case, the agent's mode is set to *DEPARTED* and its time stamp is also fast-forwarded a full period. This ensures that it does not accomplish any demands that day, and it allows the "World" class to remove it from any further simulation.

The **Support()** method is only executed by leader agents when "Transformational Leadership" is turned on in the user interface. Similar to the previous two methods, it also leverages results from another method (the **teamNeedsSupport()** method). If the method, discussed further below, returns "true" results, the **Support()** method is where the leader decides to conduct a full-team intervention to understand all of the concerns, or if it will perform a one-on-one daily support (referred to in the code as a micro-support) action. If it conducts a full-team intervention, the code assumes that it will understand who is most concerned and will adjust its leadership support accordingly.

Additionally, if "Sociometric Badge Awareness" is turned on, the code simulates researchers telling the leader that his or her team is showing concerns. The leader then conducts one-on-one daily supports with each of the team members to understand and address their concerns.

Finally, the **Complete()** method is executed when the agent is completely through its task list.

Coordination Methods. These methods govern how the agents interact with each other, and there are three of them:

1. chooseAgent()
2. callForHelp()
3. refreshAveIntNetDensity()

The **chooseAgent()** method governs how an agent chooses another agent to cooperate with. The model assumes that if an agent has to work with another agent in order to accomplish the task, then there is some aspect of being dissimilar needed. While this is not necessarily realistic in all cases (e.g., in a divide-and-conquer scenario), this method includes logic for agents to prioritize working with a random dissimilar agent first, followed by a similar agent if no dissimilar agents are available.

The **callForHelp()** method simulates coordination-required task completion and updates each agent's respective tie strengths as a result of the interaction. Note that the absolute value of the tie strength is irrelevant to the simulation. However, the nature of the trend (i.e. positive or negative) is important, as will be discussed further below. The rules governing the updated tie strengths are as follows:

- General rules

  - All interactions between agents similar in race AND gender generate a 10% tie strength adjustment to account for similarity/homophily
  - All interactions between agents similar in race OR gender generate an initial ± 5% adjustment (equal-distribution random selection) that is further moderated according to the diversity concern rules below
  - All interactions between completely dissimilar agents generate a ± adjustment (25/75 distribution biased toward -5% to simulate the conflict generated by dissimilarity) that is further moderated according to the diversity concern rules below

- Diversity concern-related rules:
  - If either agent is getting their diversity concerns addressed by the leader, the model simulates that the agents are working better together. Therefore, tie strength is only increased
  - If either agent has accumulated more than a half-threshold of diversity concerns that are not being addressed by the leader, any decrease in tie strength factor is doubled.

Finally, the last method in this category is the **refreshAveIntNetDensity()** method to update the agent's average internal network density. It is a simple method that calculates the new average value and adds it to a list for trend-tracking purposes.

Diversity Consequence Methods. These methods simulate the consequences that might be generated as a result of diverse team members working together. The model simulates these potential consequences with the following methods:

1. generateRandomDiversityConcern()
2. generateRandomDayOff()
3. generateAgentDeparture()

The **generateRandomDiversityConcern()** method simulates a team member answering a yes/no survey question once per day relating to whether or not they experienced any diversity concerns that day, and it establishes the rules guiding whether or not an agent feels concerned. These rules, which can be considered assumptions for this early model, are as follows:

- Agents with average internal network densities above 0.9 will not generate diversity concerns related to their team membership
- All other agents, regardless of race or gender, start with a 10% chance of generating a diversity concern regardless of diversity approach. This simulates the Tsui et al. (1992) research that majority members may have diversity-related concerns as well.
- Agents with internal network densities that have a positive internal network density trend periods are assumed to be 75% less susceptible to the remaining rules

- In *value in equality* diversity approaches, agents in social groups with moderate representation are more likely than their counterparts to have concerns (Apfelbaum et al., 2016). On the other hand, agents in social groups with low representation are more likely than their counterparts to have concerns when operating under *value in difference* diversity approaches (Apfelbaum et al., 2016). This difference is captured in the model by biasing the percentage of a diversity concern up from 10% to 25% for the respective social groups.

Next, as these diversity concerns accumulate, so do the chances of facing other consequences like absences and agent departures. The **generateRandomDayOff()** method simulates growing unhappiness with the firm by first checking to see if the leader is failing to address the diversity concerns, and then checking to see if the concerns have accumulated past the diversity concern threshold. If both are true, the agent randomly selects to take the day off (equal distribution between yes and no).

Finally, the **generateAgentDeparture()** simulates an agent deciding to leave the environment. Similar to many class attendance policies, the absence threshold in the simulation is currently set to 10%. Any agent missing more than this many absences then definitively departs the environment for the remainder of the simulation.

Leader Action Methods. The final set of significant methods in the "Agent" class simulates the actions of the team leader, and there are three of them:

1. teamNeedsSupport()
2. generateLeaderIntervention()
3. generateMicroSupport()

These methods are only conducted by leaders when "Transformational Leadership" is turned on in the simulation, and they each have separate sets of rules for leaders with and without sociometric badge awareness.

The **teamNeedsSupport()** method simulates an agent coming to work and taking time to check if his or her team needs any support. Without sociometric badge awareness, the model assumes the leader will not be triggered to check on the team until it notices an agent (as part of the **Support()** method described above) has been absent 50% of the threshold. With the threshold currently set at 10% of the project span, this trigger occurs if an agent is absent 5% of the total project span. The leader then recognizes that its team needs support, holds a team-wide "intervention" to ensure it understands all of the issues at play, and begins to work them off as part of its daily task list. As part of this burndown, a leader will decide to hold daily micro-supports (e.g., small conversations) with the other agents.

With sociometric badge awareness, the model simulates time required to learn from the badges and correlate associated diversity concerns. After an agent passes the 25% threshold mark, the model assumes that researchers will be able to give a leader enough information to have conversations with applicable team members.

While a leader will select from both of these activities, it also has its non-diversity-related tasks as well. During this time it is focused on those tasks, other diversity-related concerns are still allowed in order to simulate the complexity of the environment.

Next, the **generateLeaderIntervention()** method, as introduced above, triggers when a leader without sociometric badge awareness notices that an agent is gone for more than 50% of its available threshold (note - this method is not applicable to leaders with sociometric badge awareness at this time). The intervention method simulates a full-day off-site where agents do no work, but instead talk to the leader and to each other about their concerns. The method assumes the leader is skilled at facilitating these types of events in a trustworthy manner such that all agents get a boost in tie strength, and the agent that triggered the intervention gets all of its diversity concerns addressed on the spot.

Finally, the **generateMicroSupport()** method simulates daily interaction by the leader to mitigate concerns as applicable. Both with and without sociometric badge awareness, the method assumes that the leader has insights (either from the intervention or from talking with the agents) to understand who needs the most support. In this way, the method guides the burndown of the diversity concerns on the team.

4.4 Model Validation

This section builds on the model description presented in the last section by offering a few simple scenarios to ensure the model functions in a realistic enough manner to generate insights. For each of the scenarios presented herein, the following input parameters were used:

- Project Duration (Days): 28
- # of Formal Leaders: 1
- # of Engineers: 11

As introduced in Section 4.3.1, the 11-engineer team is a test team with specified demographics in accordance with the representation of each social group as described in Apfelbaum et al. (2016). Choosing a different mix of engineers does not provide as much validation confidence as the 11-engineer team.

Additionally, this section first builds confidence in the model using the single simulation user interface. Once this validation is established, the section closes with a validation that the Monte Carlo code is correctly capturing the results from single simulations.

4.4.1 Validation #1: Internal Network Density Adjustments

Goal. This mini-experiment tests the impacts of different simulation events on internal network density. Specifically, the following events are tested:

1. Project execution with no required interaction/coordination by a homogeneous team
2. Project execution with high amount of interaction by a homogeneous team

3. Project execution with no required interaction/coordination by a heterogeneous team
4. Project execution with high amount of interaction by a heterogeneous team

Expected Results. The model simulates the creation of diversity concerns as a result of agents working on a heterogeneous team in environments with diversity approaches that they do or do not prefer. Therefore, on homogeneous teams, regardless of diversity approach, there should be no diversity concerns generated and no negative impact to internal network density when interaction is not required. Further, when interaction is required, internal network density should increase as a result of homophily on homogeneous teams.

Additionally, because the model is currently limited to simulating interaction-driven internal network density changes, heterogeneous teams also should see no decrease in internal network density when interaction is not required. They should, however, see significant negative changes as coordination percentage is increased due to the diversity approach chosen and their respective diversity-approach preferences (Apfelbaum et al., 2016).

Simulation Setup. This test uses the following 4 simulation setups:

1. This portion executes the simulation three times, each with a different homogeneous 11-member team (highly-represented agents, moderately-represented agents, and low-represented agents) executing a project that does not require interaction. Further, the setup includes the diversity approach that each respective social group does not prefer in order to ensure it is not driving non-interaction-driven diversity concerns. While different levels of representation are not applicable in homogeneous teams, this test variation is needed to ensure that the demographics of agents represented in each respective social group respond accordingly to the chosen diversity approach.
2. This test conducts the same set of simulations as the first test, but adjusts the task coordination factor to 100%.
3. This test adjusts the setup to an 11-member heterogeneous team that executes a project with no required interaction. As a heterogeneous team, only varying the diversity approach is sufficient to validate internal network density DOES NOT change throughout the projects.
4. This test conducts the same set of simulations as the previous test, but adjusts the task coordination factor to 100%.

Results. All tests revealed expected results, as shown in Table 4.1. Additionally, sample screen shots of each test are provided in Figures 4-4, 4-5, 4-6, and 4-7, respectively.

Test #	Team Demographics	Diversity Approach	Coordination Factor (%)	Starting IND	Final IND	Expected Results
1	Homogeneous / High-Rep	Equality	0	0.86	0.86	100% Met
2	Homogeneous / Mod-Rep	Equality	0	0.87	0.87	100% Met
3	Homogeneous / Low-Rep	Difference	0	0.83	0.83	100% Met
4	Homogeneous / High-Rep	Equality	100	0.84	0.93	100% Met
5	Homogeneous / Mod-Rep	Equality	100	0.86	0.94	100% Met
6	Homogeneous / Low-Rep	Difference	100	0.85	0.93	100% Met
7	Heterogeneous	Equality	0	0.77	0.77	100% Met
8	Heterogeneous	Difference	0	0.77	0.77	100% Met
9	Heterogeneous	Equality	100	0.81	0.45	100% Met
10	Heterogeneous	Difference	100	0.77	0.43	100% Met

Table 4.1: Validation Test #1 Results

Figure 4-4: Validation Test #1a: Homogeneous Team With 0% Interaction *(Value in Equality)*

Figure 4-5: Validation Test #1b: Homogeneous Team With 100% Interaction *(Value in Difference)*

Figure 4-6: Validation Test #1c: Heterogeneous Team With 0% Interaction *(Value in Equality)*

Figure 4-7: Validation Test #1d: Heterogeneous Team With 100% Interaction *(Value in Difference)*

4.4.2 Validation #2: Diversity-Concern Generation

Goal. This mini-experiment tests diversity concern generation as moderated by the diversity approach the agents are operating in.

Expected Results. Per the Apfelbaum et al. (2016) study, agents in social groups with low representation should generate more concerns in *value in difference* approaches, whereas agents in moderate social groups should generate more concerns in *value in equality* approaches

Simulation Setup. Two iterations of this test are needed, and each are done with ten runs. The first tests the *value in equality* approach, and the second tests the *value in difference approach.* "Transformational Leadership" and "Sociometric Badge Awareness" are both turned off in each simulation. Also, in order to build confidence in the model and understand the scope of concerns generated, the options to take the day off or depart are commented out of the code during these initial simulations.

Results. Both tests had expected results and are presented in Table 4.2 and Table 4.3, respectively.

Test #	High-Rep Concerns	Mod-Rep Concerns	Low-Rep Concerns	Expected Results
1	0	11	0	100% Met
2	1	9	0	100% Met
3	2	9	0	100% Met
4	0	9	0	100% Met
5	0	10	1	100% Met
6	1	13	0	100% Met
7	0	7	0	100% Met
8	0	13	0	100% Met
9	1	15	1	100% Met
10	1	16	0	100% Met

Table 4.2: Validation Test #2a Results *(Value in Equality)*

Test #	High-Rep Concerns	Mod-Rep Concerns	Low-Rep Concerns	Expected Results
1	2	1	5	100% Met
2	1	0	10	100% Met
3	2	0	11	100% Met
4	2	3	11	100% Met
5	0	0	7	100% Met
6	1	0	9	100% Met
7	2	2	11	100% Met
8	2	0	8	100% Met
9	1	0	12	100% Met
10	1	0	8	100% Met

Table 4.3: Validation Test #2b Results *(Value in Difference)*

4.4.3 Validation #3: Random Day Off Generation

Goal. This mini-experiment tests to see if agents randomly take the day off if their diversity concerns accumulate beyond the diversity consequence threshold.

Expected Results. The diversity consequence threshold is established at 10% of the project span. That is, if agents feel concerned more than 10% of the days, they will consider taking the day off. For a 28-day project, the consequence threshold is 3 days. Agents will not be taking the day off with less than 3 diversity concerns generated, but they may have more than 3 concerns and no days off if they elect to come to work.

Simulation Setup. Two iterations of this test are needed in order to stress both low- and moderately-represented social groups. As in Validation #2, each are done with ten runs. The first tests the *value in equality* approach, and the second tests the *value in difference approach.* "Transformational Leadership" and "Sociometric Badge Awareness" are both turned off in each simulation. Additionally, similar to Validation

Test #2, the ability for agents to depart after reaching the absence threshold is turned off for this validation test.

Results. Both tests had expected results and are presented in Table 4.4 and Table 4.5, respectively.

Test #	Concern Threshold Met	Agent Absences			Expected Results
		High-Rep	Mod-Rep	Low-Rep	
1	Yes	0	15	0	100% Met
2	Yes	0	27	0	100% Met
3	Yes	0	27	0	100% Met
4	Yes	0	9	0	100% Met
5	Yes	0	25	0	100% Met
6	Yes	0	16	0	100% Met
7	Yes	0	27	0	100% Met
8	Yes	0	30	0	100% Met
9	Yes	0	9	0	100% Met
10	Yes	0	20	0	100% Met

Table 4.4: Validation Test #3a Results *(Value in Equality)*

Test #	Concern Threshold Met	Agent Absences			Expected Results
		High-Rep	Mod-Rep	Low-Rep	
1	Yes	0	0	18	100% Met
2	Yes	0	0	6	100% Met
3	Yes	0	0	14	100% Met
4	Yes	0	0	11	100% Met
5	Yes	0	0	4	100% Met
6	Yes	0	0	8	100% Met
7	Yes	0	0	3	100% Met
8	Yes	0	0	19	100% Met
9	Yes	0	0	11	100% Met
10	Yes	0	0	6	100% Met

Table 4.5: Validation Test #3b Results *(Value in Difference)*

4.4.4 Validation #4: Agent Departure Generation

Goal. This mini-experiment tests to see if agents depart after their diversity concern and absence thresholds are met.

Expected Results. The absence threshold is established at 10% of the project span. That is, if agents are inactive for more than 10% of the days, they will permanently depart the simulation. For a 28-day project, the absence threshold is 3 days. Agents will not depart with less than 3 absences, nor will they stay to collect more than 3 absences.

Simulation Setup. Two iterations of this test are needed in order to stress both low- and moderately-represented social groups. As with the previous validations, each are done with ten runs. The first tests the *value in equality* approach, and the second tests the *value in difference approach.* "Transformational Leadership" and "Sociometric Badge Awareness" are both turned off in each simulation. However, the code associated with days off and agent departures is activated for these simulations.

Results. Both tests had expected results and are presented in Table 4.6 and Table 4.7, respectively.

Test	Absence Threshold		Agent Departures			Expected
#	Met	Exceeded	High-Rep	Mod-Rep	Low-Rep	Results
1	Yes	No	0	2	0	100% Met
2	Yes	No	0	2	0	100% Met
3	Yes	No	0	1	0	100% Met
4	Yes	No	0	2	0	100% Met
5	Yes	No	0	1	0	100% Met
6	Yes	No	0	3	0	100% Met
7	Yes	No	0	1	0	100% Met
8	Yes	No	0	3	0	100% Met
9	Yes	No	0	2	0	100% Met
10	No	No	0	0	0	100% Met

Table 4.6: Validation Test #4a Results *(Value in Equality)*

Test	Absence Threshold		Agent Departures			Expected
#	Met	Exceeded	High-Rep	Mod-Rep	Low-Rep	Results
1	Yes	No	0	0	1	100% Met
2	No	No	0	0	0	100% Met
3	Yes	No	0	0	1	100% Met
4	Yes	No	0	0	2	100% Met
5	Yes	No	0	0	2	100% Met
6	Yes	No	0	0	2	100% Met
7	Yes	No	0	0	1	100% Met
8	Yes	No	0	0	2	100% Met
9	Yes	No	0	0	2	100% Met
10	Yes	No	0	0	1	100% Met

Table 4.7: Validation Test #4b Results *(Value in Equality)*

4.4.5 Validation #5: Transformational Leadership Support Behaviors

Goal. This mini-experiment tests to see if leader agents are triggered by absences to address diversity concerns among the agents.

Expected Results. A leader agent will meet with the entire team when an agent passes one-half of its absence threshold, which for a 28-day project is two days. The

leader will then begin addressing the diversity concerns throughout the team. Also expected is an increase in internal network density.

Simulation Setup. Two iterations of this test are needed in order to stress both low- and moderately-represented social groups. As with the previous validations, each are done with ten runs. The first tests the *value in equality* approach, and the second tests the *value in difference approach.* "Transformational Leadership" is turned on for these simulations, but "Sociometric Badge Awareness" is still turned off.

Results. Both tests had expected results and are presented in Table 4.8 and Table 4.9, respectively. Further, Figure 4-8 provides a sample output of a Validation #5b simulation.

Test #	Interventions Needed	Absence Trigger	Further Concerns Addressed?	Initial Int Net Density	Final Int Net Density	Expected Results
1	1	2	Yes	0.78	0.89	100% Met
2	1	2	Yes	0.79	0.87	100% Met
3	1	2	Yes	0.78	0.90	100% Met
4	2	2	Yes	0.82	0.91	100% Met
5	1	2	Yes	0.80	0.91	100% Met
6	1	2	Yes	0.79	0.88	100% Met
7	1	3	Yes	0.78	0.88	100% Met
8	1	2	Yes	0.79	0.80	100% Met
9	1	2	Yes	0.81	0.88	100% Met
10	1	2	Yes	0.78	0.80	100% Met

Table 4.8: Validation Test #5a Results *(Value in Equality)*

Test #	Interventions Needed	Absence Trigger	Further Concerns Addressed?	Initial Int Net Density	Final Int Net Density	Expected Results
1	1	2	Yes	0.78	0.87	100% Met
2	1	3	Yes	0.81	0.89	100% Met
3	1	3	Yes	0.85	0.90	100% Met
4	0	N/A	N/A	0.78	0.78	100% Met
5	0	N/A	N/A	0.81	0.87	100% Met
6	1	2	Yes	0.81	0.89	100% Met
7	1	2	Yes	0.77	0.85	100% Met
8	1	2	Yes	0.80	0.86	100% Met
9	1	2	Yes	0.78	0.82	100% Met
10	1	2	Yes	0.77	0.81	100% Met

Table 4.9: Validation Test #5b Results *(Value in Difference)*

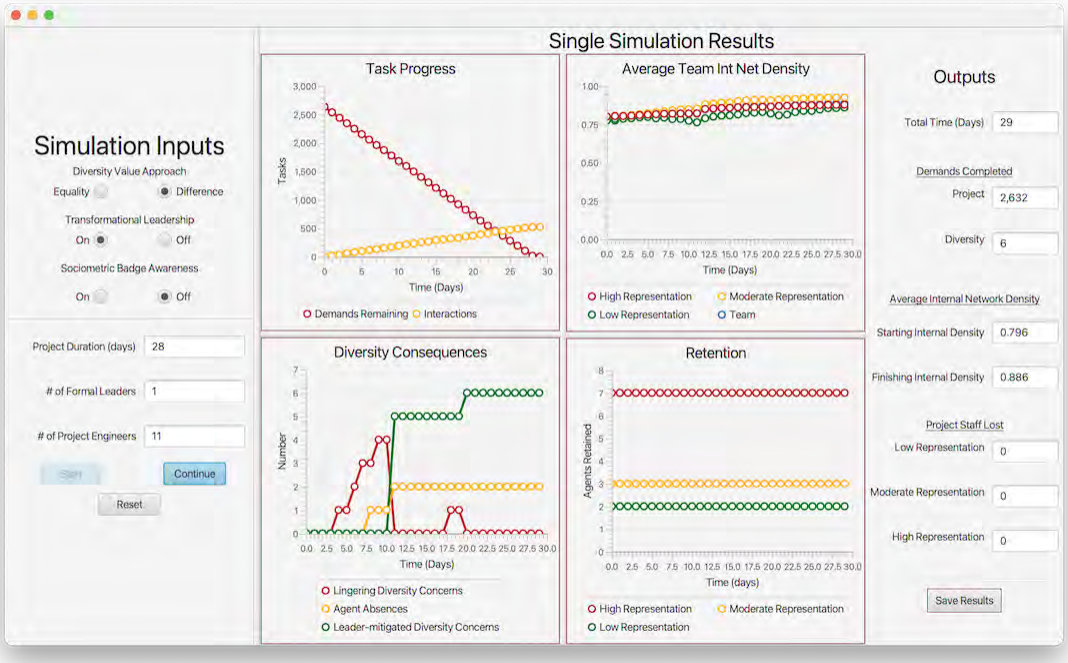

Figure 4-8: Sample Simulation Output - Validation #5b

4.4.6 Validation #6: Sociometric Badge Awareness

Goal. This mini-experiment tests to see if leader agents support the engineering agents with insights from sociometric badges.

Expected Results. A leader agent will get anonymous insights from sociometric badges (presumably from researchers), which will in turn trigger daily support actions in accordance with the Amabile et al. (2004) study. With daily tasks that are not diversity related, the leader will need to decide if it will support the team members or if it will work on its other tasks. As in Validation #5a, internal network density should typically increase. Finally, there should be no need for interventions with daily team member support.

Simulation Setup. Two iterations of this test are needed in order to stress both low- and moderately-represented social groups. As with the previous validations, each are done with ten runs. The first tests the *value in equality* approach, and the second tests the *value in difference approach.* "Transformational Leadership" and "Sociometric Badge Awareness" are both turned on for these simulations.

Results. Both tests had expected results and are presented in Table 4.10 and Table 4.11, respectively, and Figure 4-9 illustrates a sample Validation #5a simulation.

Test #	Concerns Addressed?	Initial Int Net Density	Final Int Net Density	Expected Results
1	Yes	0.77	0.88	100% Met
2	Yes	0.77	0.87	100% Met
3	Yes	0.80	0.87	100% Met
4	Yes	0.80	0.88	100% Met
5	Yes	0.77	0.88	100% Met
6	Yes	0.81	0.88	100% Met
7	Yes	0.82	0.91	100% Met
8	Yes	0.79	0.89	100% Met
9	Yes	0.81	0.88	100% Met
10	Yes	0.81	0.91	100% Met

Table 4.10: Validation Test #6a Results *(Value in Equality)*

Test #	Concerns Addressed?	Initial Int Net Density	Final Int Net Density	Expected Results
1	Yes	0.80	0.89	100% Met
2	Yes	0.82	0.89	100% Met
3	Yes	0.78	0.88	100% Met
4	Yes	0.78	0.88	100% Met
5	Yes	0.76	0.88	100% Met
6	Yes	0.78	0.88	100% Met
7	Yes	0.80	0.89	100% Met
8	Yes	0.74	0.89	100% Met
9	Yes	0.79	0.90	100% Met
10	Yes	0.80	0.90	100% Met

Table 4.11: Validation Test #6b Results *(Value in Difference)*

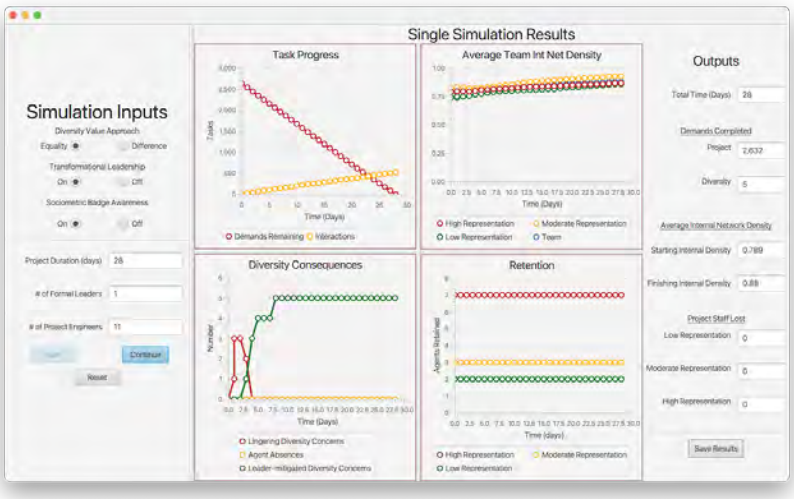

Figure 4-9: Sample Simulation Output - Validation #6a

4.4.7 Validation #7: Monte Carlo Simulation Operation

Goal. This test validates the Monte Carlo code correctly captures results from multiple consecutive single simulation operations.

Expected Results. The results presented in the charts and the Microsoft Excel® output file are expected to be consistent with previously presented results.

Simulation Setup. One iteration of this test is needed to ensure the code is capturing multiple simulation results. Specifically, the test will use the Transformational Leadership code as it will capture all functionality required across the other scenario types. The test will also use a *value in equality* diversity approach to generate the greatest number of concerns. The simulation will include 10 individual runs, and the final internal network density of each run will be compared with results presented in Table 4.8.

Results. The test had results in family with the previously-presented results, as expected. The final internal network density values are captured in Table 4.12. Additionally, selected screen shots of the simulation results are presented in Figure 4-10 for illustrative purposes. Finally, the code correctly captured the following data points for each simulation in the output Microsoft Excel® files:

1. Number of Interactions
2. Starting Internal Network Density
3. Final Internal Network Density
4. Generated Diversity Concerns
5. Absences
6. Leader Interventions
7. Leader Daily Supports
8. Agents Lost (Low, Mod, High)
9. Project Duration

Test #	Table 4.8 Results			Interventions Needed	Further Concerns Addressed?	Final Int Net Density
	Interventions Needed	Further Concerns Addressed	Final Int Net Density			
1	1	Yes	0.89	1	Yes	0.86
2	1	Yes	0.87	0	Yes	0.78
3	1	Yes	0.90	1	Yes	0.87
4	2	Yes	0.91	1	Yes	0.82
5	1	Yes	0.91	1	Yes	0.85
6	1	Yes	0.88	3	Yes	0.87
7	1	Yes	0.88	1	Yes	0.86
8	1	Yes	0.80	2	Yes	0.92
9	1	Yes	0.88	1	Yes	0.91
10	1	Yes	0.80	2	Yes	0.93
Result: Monte Carlo results are in family with Table 4.8						

Table 4.12: Validation Test #7a Results *(Value in Equality)*

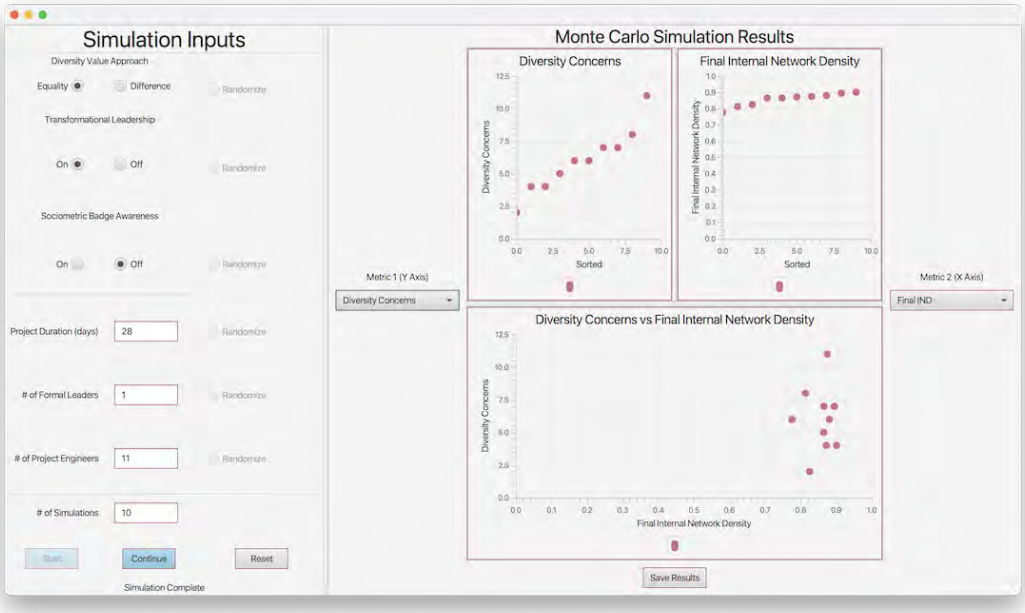

Figure 4-10: Monte Carlo Validation Simulation *(Value in Equality)*

4.5 Chapter Summary

This chapter discussed an agent-based model built in response to the research question and hypothesis presented in Chapter 3. Specifically, it consolidated the assumptions used, provided an introduction to the model, and then discussed the details of each of its significant functions. It then discussed the results of the model's validation program. The next chapter presents the model's simulation results of the proposed research experiment in Section 3.3.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Discussion of Results

This chapter presents and discusses this research effort's simulation results. As introduced in Chapter 4, the simulated scenario is a 90-day project with an 11-member engineering team and a single formal team leader. The chapter begins with results from base case simulations (i.e., no transformational leadership or sociometric badge awareness) to provide the foundation for comparison with simulations that introduce the additional leadership actions. Next, results of simulations with transformational team leaders are presented, followed by the results of simulated sociometric badge experiments. Further, each scenario is simulated under varying diversity approaches.

Each section first presents the results of the respective 2000-run Monte Carlo simulations. Using a Monte Carlo simulation approach, the resulting metrics of each run can be collectively pooled to highlight general trends and insights across multiple teams. Next, discussions of sample single simulation results then amplify the Monte Carlo results with insights in to the scenarios' daily dynamics. Each of these discussions include insights from comparisons with previously-presented results.

It is important to note here, however, that the values of the results presented in this section are not to be interpreted absolutely. The initial version of the agent-based simulation code needs further research and empirical validation efforts followed by updates for improved precision. Instead, the results are presented here in order that they can be interpreted in comparison with other simulation results.

Finally, note that there will be two types of plots presented in this section. The first type includes screen shots of both single and Monte Carlo simulation results from the agent-based model. The other type are data visualizations using Python code. These visualizations are not part of the agent-based model, but provide important insights for further consideration.

5.1 Base Case Simulation Results

Given the nature of this research interest in teams operating in either *value in equality* or *value in difference* diversity approaches, two base case scenarios provide the initial data for further simulation comparison. Specifically, these scenarios simulate teams with leaders that do not perform any transformational leadership behaviors, and

therefore by extension, do not have the opportunity to take advantage of sociometric badge awareness.

5.1.1 Monte Carlo Results

Two 2000-run Monte Carlo simulations represent the base case scenarios:

- *Value in equality* diversity approach, **without** transformational leadership and **without** sociometric badge awareness
- *Value in difference* diversity approach, **without** transformational leadership and **without** sociometric badge awareness

The Monte Carlo simulations record data at the end of each run. This section presents the simulation metrics in correlation with one another. The correlation matrices in Table 5.1 and Table 5.2 present the results of the *value in equality* and *value in difference* base case Monte Carlo simulations, respectively.

	1	2	3	4	5	6
1. Interactions	1					
2. Internal Net Density	-0.031	1				
3. Diversity Concerns	0.000	-0.88	1			
4. Absences	0.002	-0.52	0.79	1		
5. Turnover	-0.002	-0.36	0.63	0.88	1	
6. Project Duration	-0.012	-0.38	0.63	0.85	0.88	1

Table 5.1: Correlation Matrix: Base Case Monte Carlo Simulation *(Value in Equality)*

	1	2	3	4	5	6
1. Interactions	1					
2. Internal Net Density	-0.026	1				
3. Diversity Concerns	-0.011	-0.80	1			
4. Absences	-0.018	-0.48	0.81	1		
5. Turnover	-0.028	-0.26	0.63	0.85	1	
6. Project Duration	-0.00	-0.30	0.61	0.79	0.84	1

Table 5.2: Correlation Matrix: Base Case Monte Carlo Simulation *(Value in Difference)*

The remainder of this section discusses each metric with respect to its most-highly correlated pair.

Correlation of Interactions & Internal Network Diversity. The correlation results show that the overall number of interactions was not significantly related to any other metric. Contrary to the system dynamics model proposed in Figure 3-4, this includes internal network density. Instead, the data (see Figure 5-1 and Figure 5-2) reveal that approximately 1,500 of the runs result in internal network densities

lower than 0.75, with the remaining teams ending with stronger ties. Some teams even attained tie strengths above 0.9.

It is important to note here, however, that there is no optimization code included in this initial model to drive agents towards optimizing their internal strength. Instead, the interaction-driven changes in internal network density are an emergent result of the *callForHelp()* method's rules described in Section 4.3.4. Specifically, demographic affinity rules govern the majority of changes in tie strength with each interation. Additionally, as some research indicates increased interactions can improve team performance (Horwitz and Horwitz (2007); Kearney and Gebert (2009)), while other research indicates that the improvements are not quite enough to overcome the negative effects brought on by interpersonal conflict (Pelled et al. (1999); Ancona and Caldwell (1992)), the model also includes a randomized contribution to the agent-pair's tie strength. Together, these contributions are additive over the course of the project and result in the final average internal network density.

Of the two base cases, the *value in equality* approach has a broader range of final internal network densities (approximately 0.41 - 0.94) than that of the *value in difference* approach (approximately 0.48 - 0.94). The former also resulted in more simulations at the high end (approximately 250 above 0.8, for example) than the latter (approximately 200 above 0.8), as well. This could be an indicator that *value in equality* approaches may hold more risk to overall performance. That is, as the *value in equality* approach more negatively affects the higher population of moderately-represented social groups, that higher population has a stronger affect on final overall internal network density. As more people have more positive interactions, they can achieve more positive final results (and vice versa).

Figure 5-1: Base Case Monte Carlo Results: Internal Network Density & Interactions *(Value in Equality)*

Figure 5-2: Base Case Monte Carlo Results: Internal Network Density & Interactions *(Value in Difference)*

Correlation of Internal Network Density & Diversity Concerns. The correlation of results in both base case scenario simulations reflect a strong negative correlation between internal network density and diversity concerns (see Figure 5-3 and Figure 5-4). That is, as internal network density increases, the number of diversity concerns decrease. This is not an emergent quality of the model, but rather an expected result as the *generateRandomDiversityConcern()* method contains a rule that negative internal density trends are more likely to lead to increased diversity-related concerns. Although it is not emergent, it represents an important aspect of the model's operation in accordance with previous research.

Another expected result is that there were some teams that generated zero / near-zero concerns. This happens in the model when all members of the team start the project with strong average internal network densities (which is also randomized in the model). That is, model results reflect that agents with strong initial ties with each other are more likely to maintain that bond, thus preventing diversity-related concerns from surfacing in the first place.

There does appear to be a notable emergent difference between the differing diversity approaches that manifests here, however. Specifically, some teams operating in *value in equality* generated larger numbers of concerns than their counterparts with the largest number of concerns in the *value in difference* simulation. Similar to the previous correlation discussion, and in accordance with research that shows *value in equality* approaches are less preferred by groups with more moderate representation (Apfelbaum et al., 2016), I expect that the higher number of agents on these teams could generate more overall concerns than their counterparts in the opposite situation.

In addition to this expectation, however, these charts reveal that leaders may face a more daunting challenge to keep tie strengths up when *value in equality* approaches are used.

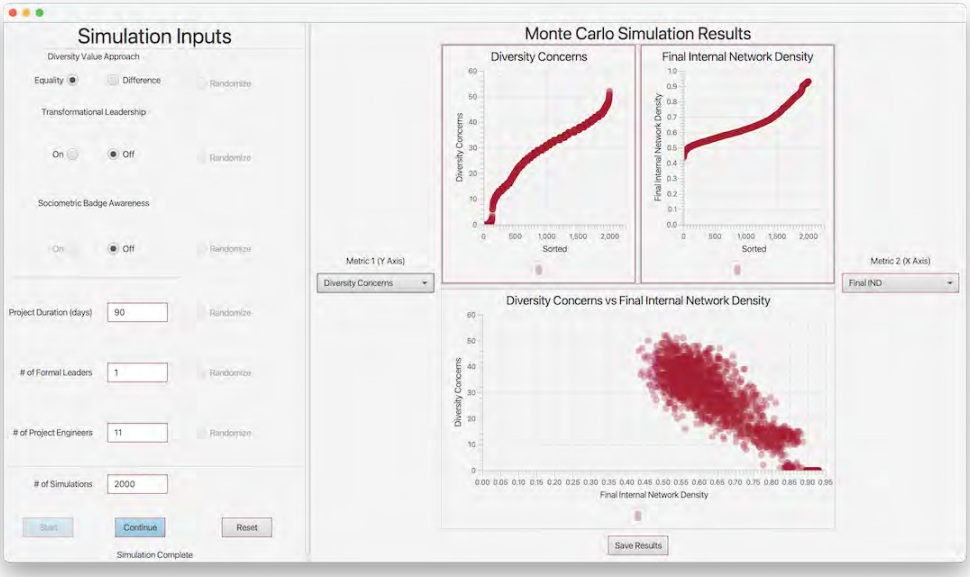

Figure 5-3: Base Case Monte Carlo Results: Internal Network Density & Diversity Concerns *(Value in Equality)*

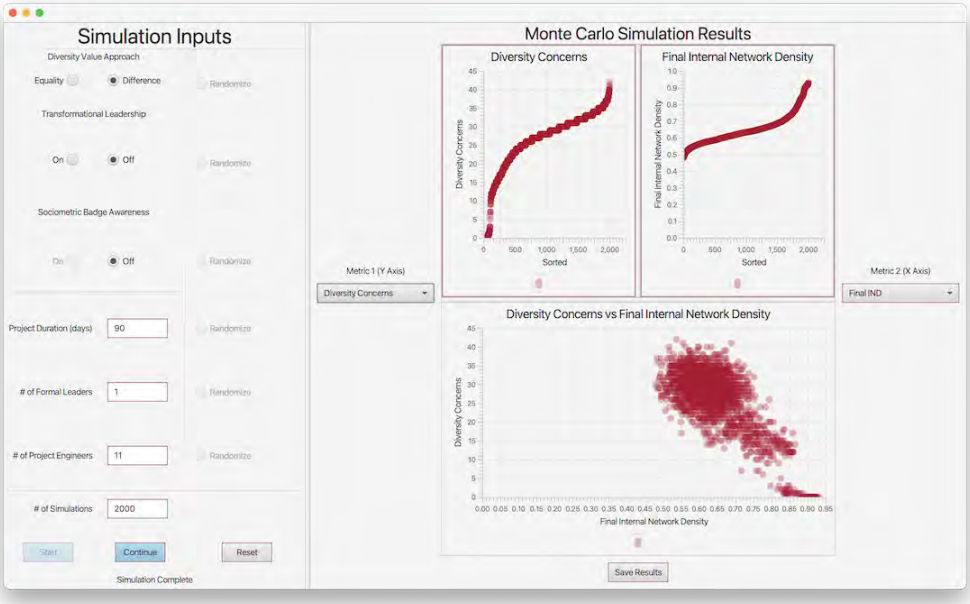

Figure 5-4: Base Case Monte Carlo Results: Internal Network Density & Diversity Concerns *(Value in Difference)*

Correlation of Agent Absences & Diversity Concerns. Diversity concerns are most strongly correlated with agent absences, and as expected, the simulation results show a positive relationship. Moreover, the shape of the data (see Figure 5-5 and Figure 5-6) show a stair step shape that, while not intentionally coded in the model, results from the threshold that removes agents if they are absent more than 10% of the project duration. With a 90-day project, agents are removed at the 10-absence point. The stair step shape, therefore, also hints at the agent turnover in each simulation run. That is, runs that lost 1 agent are the prominent lower step, and each successive step represents another agent lost in the simulation.

Along these lines, another significant difference between the two base cases is in the number of agents lost. Similar to the results seen in the previous sections, moderately-represented social groups lost more agents in *value in equality* approaches because they had more agents to lose.

The results presented above show the *value in equality* approach may disadvantage most teams; however, these charts hint at a potential advantage. Corresponding to the larger number of generated diversity concerns overall, the *value in equality* simulation results show a more shallow-sloped distribution of absence results than those of the *value in difference* simulation. They also indicate a fewer number of runs that reach the maximum number of absences than *value in difference* approaches. In other words, while the *value in equality* approach shows a higher number of concerns and has the potential to lose a larger number of agents, its ability to spread those concerns across that larger number of agents appears to offer a slight resilience over *value in difference* approaches.

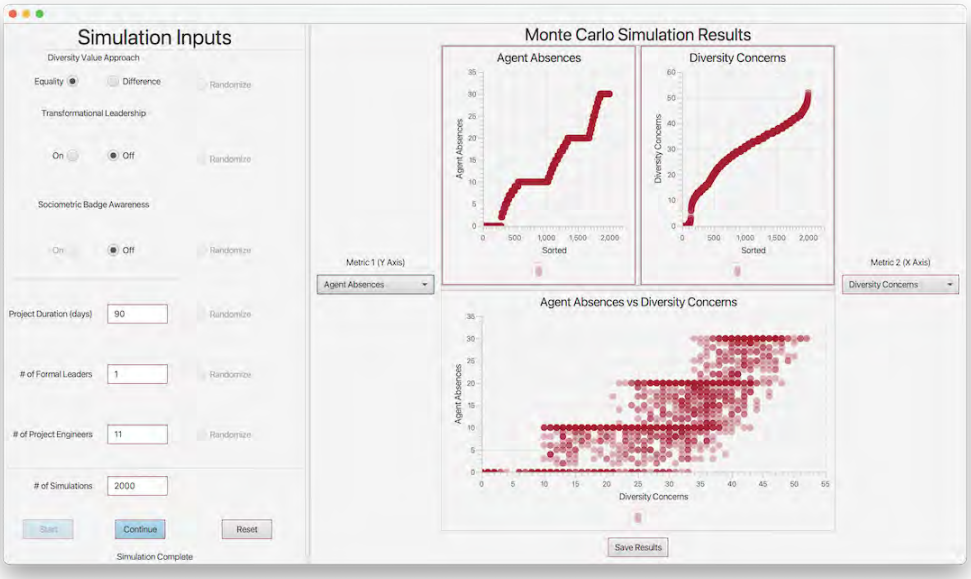

Figure 5-5: Base Case Monte Carlo Results: Diversity Concerns & Agent Absences (*Value in Equality*)

Figure 5-6: Base Case Monte Carlo Results: Diversity Concerns & Agent Absences *(Value in Difference)*

Correlation of Agent Absences & Agent Departures. In accordance with the model rules, agent absences are strongly correlated with agent departures. As previously discussed, the model currently does not include any flexibility associated with agent departures if absences grow beyond 10% of the project. As shown in Figures 5-7 and 5-8, the results confirm that some agents departed the simulation run if they reached their absence threshold.

Somewhat surprisingly, however, these results show some teams that had higher numbers of absences that correspond to lower turnover numbers. In accordance with the slight *value in equality* resilience comment above, this occurs more with *value in equality* approaches as these teams can afford to absorb more absences spread over more team members without risking turnover.

Another surprise revealed by these results is that neither base case scenario resulted in any agent departures from social groups outside of the one known to not prefer the tested diversity approach. That is, the *value in equality* simulation resulted only in agent departures from moderately represented social groups, and the *value in difference* scenario only resulted in agent departures from low-represented social groups. Though this is consistent with the Apfelbaum et al. (2016) results, as well as the validation results shown in Table 4.4 and Table 4.5, it may not be realistic. Indeed, Apfelbaum et al. (2016) note that their theory needs further application to other work environments. Accordingly, further study among technical firms is needed to validate the realism of these simulation results.

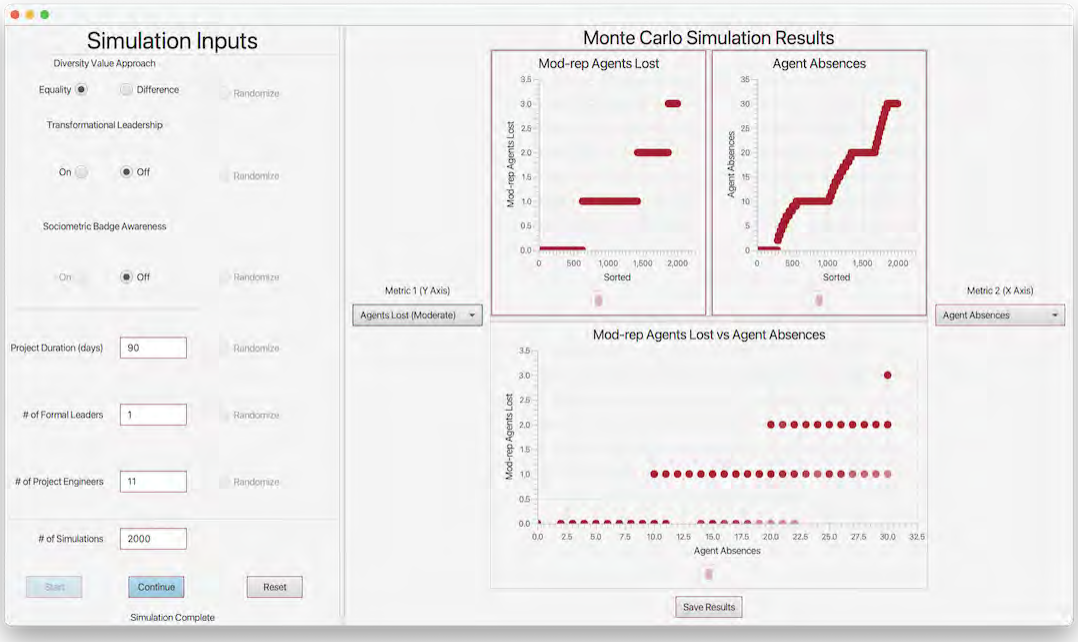

Figure 5-7: Base Case Monte Carlo Results: Agent Departures & Agent Absences *(Value in Equality)*

Figure 5-8: Base Case Monte Carlo Results: Agent Departures & Absences *(Value in Difference)*

Correlation of Agent Departures & Project Delays. As expected and self-explanatory, agent departures were strongly correlated with project delays for both base case scenarios. Further, with more agents to lose, the *value in equality* teams revealed greater project delays than their counterparts. Figures 5-9 and 5-10 illustrate this positive correlation.

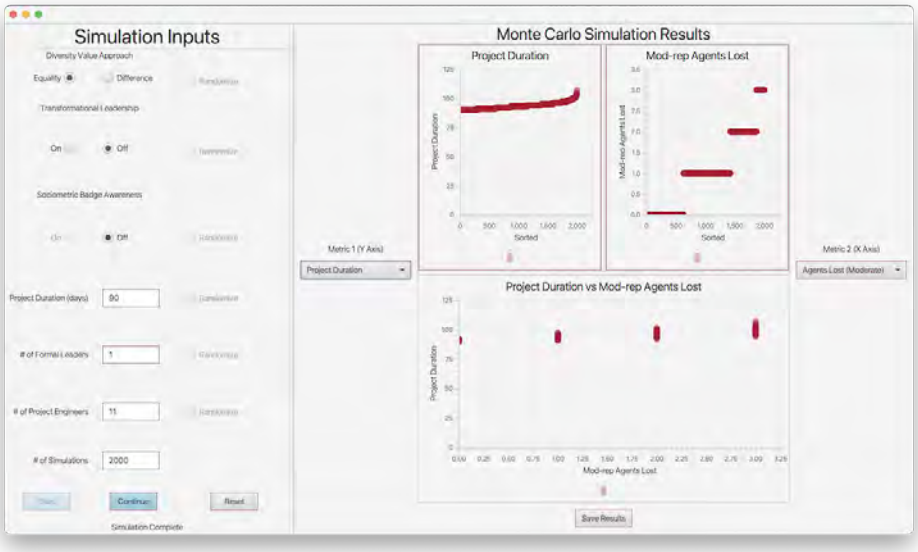

Figure 5-9: Base Case Monte Carlo Results: Project Delays & Agent Departures *(Value in Equality)*

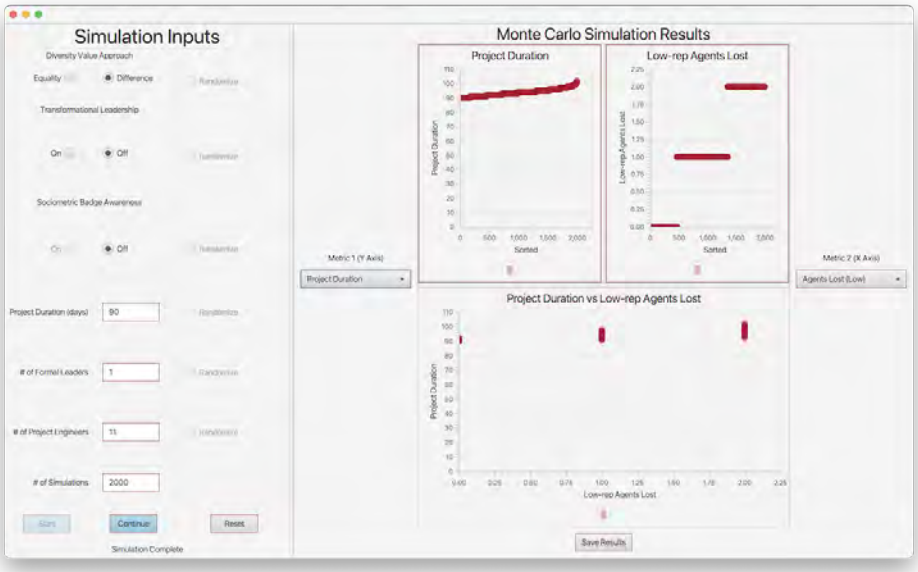

Figure 5-10: Base Case Monte Carlo Results: Project Delays & Agent Departures *(Value in Difference)*

5.1.2 Single Simulation Results

Next, single simulations provide additional insight into the team with a closer view of its daily dynamics. This section discusses the results of a single sample simulation for each scenario base case.

***Value in Equality* Base Case Scenario.** Figure 5-11 shows a sample run of the *value in equality* base case scenario. In this run, the team begins with a relatively high average internal network density, but it masks the relatively lower tie strength felt by the low-representation agents. It then begins to dwindle as agents start interacting together. Additionally, the agents are showing diversity concerns at the beginning of the project that continue to grow with increasing interactions. Those concerns eventually lead to absences, which in turn leads to two of the moderately-represented agents quitting.

By reviewing the daily dynamics of this particular team of agents, a number of important insights surface that are not otherwise seen in the Monte Carlo simulation results. That is, rather than just capturing the endpoint of each data series in the Monte Carlo results, this single simulation shows the path the agents took to get to the end point. For example, we can see the growing diversity concerns, when the absences begin to occur, what the members of each social group are "reporting" for internal network density, and when the agents departed.

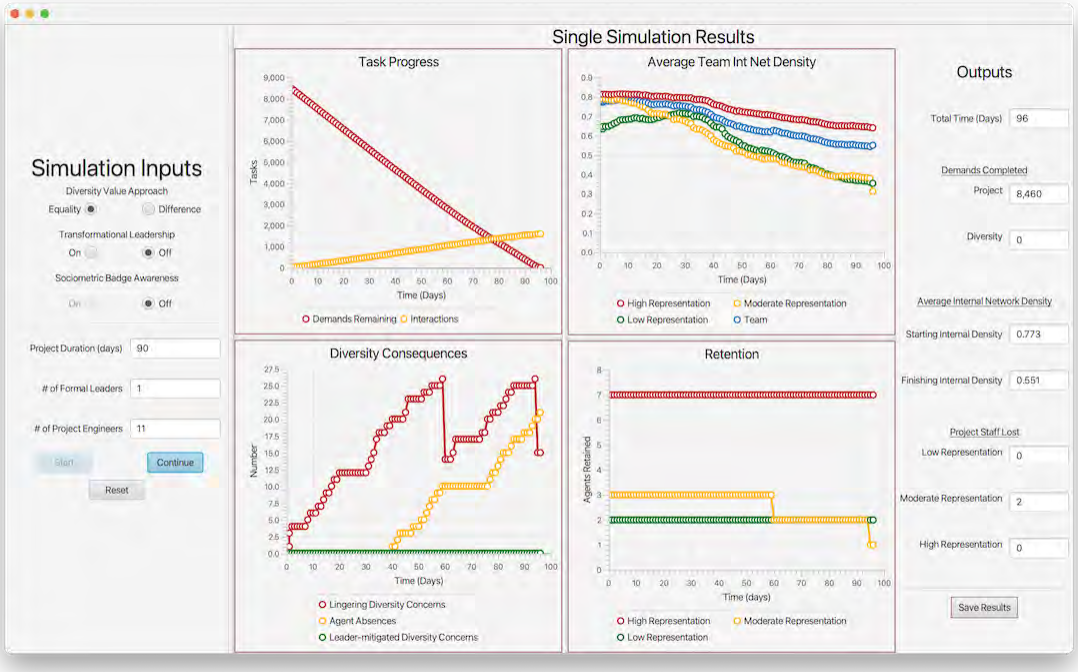

Figure 5-11: Base Case Single Simulation *(Value in Equality)*

***Value in Difference* Base Case Scenario.** Building on the previous discussion, Figure 5-12 presents another sample base case scenario simulation, but with a *value in difference* diversity approach. Under this type of approach, we would expect to see members of low-represented social groups generate more diversity concerns than the other groups, which indeed is what the simulation shows.

Additionally, another significant insight offered by these results is that the number of diversity concerns on the team drops significantly when the first agent leaves. This indicates that one low-represented agent in particular was feeling the brunt of pain, and that that pain can be quickly transferred to the remaining low-represented agent as well. After the second and final low-representation agent leaves, lingering concerns reflect that more than just these two agents on this team had concerns.

Finally, we see in this run that when the first agent departed, its social group's average internal network density noticeably increased, indicating that the other member felt a somewhat stronger, though still low, tie to the group.

Though these particular insights are not genearalizable, the important point to capture here is that the daily dynamics may be important to research efforts, yet they are not captured in typical Input-Process-Output (I-P-O)-style studies.

Figure 5-12: Base Case Single Simulation *(Value in Difference)*

5.2 Transformational Leadership Simulation Results

Based on past diversity-related research, the two base case scenarios presented in Section 5.1 clearly indicate a need to mitigate diversity concerns, prevent turnover,

and preserve project schedule. This section builds on those results by comparing them with simulated performance of teams led by transformational leaders that perform support-related behaviors for their teams.

5.2.1 Monte Carlo Results

This section again uses the Monte Carlo model to simulate 2000 teams led by transformational leaders that are triggered by increased absences to practice supportive team interventions and follow-up daily support behaviors. In general, many trends are similar to those discussed in Section 5.1, with some notable differences.

For example, the simulation continued the following trends:

- The number of interactions was insignificantly correlated with other metrics recorded in the simulation
- Internal network density was most significantly (and negatively) correlated with the number of diversity concerns the teams generated
- Diversity concerns were most significantly (and positively) correlated with the number of agent absences

New to this simulation, however, is data associated with leader interventions and daily leader behaviors that supported team members. These new data exhibited the following trends:

- Leader interventions were most significantly (and positively) correlated to the number absences on the team. As expected, leader interventions also had a strong positive correlation with project duration, thus reflecting that addressing employee concerns indeed takes time.
- Daily supports were most significantly (and positively) correlated to the number of generated diversity concerns, thus reflecting the leader's ability to address the agent concerns. However, this correlation between diversity concerns and daily supports is not 1.0, indicating that the leaders did not address all concerns in every run of the simulation. This is coded in the model as a choice the leader gets to make if the number of diversity concerns are below 25% of the diversity concern threshold.

These trends are reflected in the correlation matrix presented in Table 5.3.

	1	2	3	4	5	6	7	8
1. Interactions	1							
2. Internal Net Density	-0.043	1						
3. Diversity Concerns	-0.020	-0.66	1					
4. Absences	-0.002	-0.18	0.59	1				
5. Interventions	0.001	-0.17	0.59	0.85	1			
6. Daily Supports	-0.045	-0.43	0.89	0.50	0.50	1		
7. Agents Lost	N/A	N/A	N/A	N/A	N/A	N/A	1	
8. Project Duration	–	-0.26	0.61	0.77	0.85	0.54	N/A	1

Table 5.3: Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Equality)*

Correlation of Interactions & Internal Network Density. As in the base case, the simulation showed no general trend between the number of interactions and internal network density. Further, by comparing previously-discussed data shown in Figure 5-1 with the new results shown in Figure 5-13, we can see a small rise in the range of internal network density slightly as a result of the addition of transformational leadership support behaviors. While the floor of the base case simulations are between 0.4 and 0.5, the floor of teams led by transformational leaders in both diversity approaches rose to 0.5.

Figure 5-13: Transformational Leadership Monte Carlo Simulation Results: Internal Network Density Vs. Interactions *(Value in Equality)*

Figure 5-14: Transformational Leadership Monte Carlo Simulation Results: Internal Network Density Vs. Interactions *(Value in Equality)*

Correlation of Internal Network Density & Diversity Concerns. Similar to the base case results, the number of diversity concerns rose as internal network density dropped. However, there was a significant difference in the shape of the old data (see Figure 5-1 or Figure 5-2) and the new data (see Figure 5-15 and 5-15).

Specifically, the shapes of the base case data sets are thick, diagonal lines aimed at the (1,0) corner of the chart. With transformational leaders on the team, however, the shape of the line gets taller on the top edge and appears to expand towards a hypothetical vertical drawn at 0.95 on the x-axis. This shape shift suggests that transformational leadership may be associated with higher values of diversity concerns even though the team has a higher internal network density. This is an emergent quality of the model that provides two important insights.

First, with transformational leadership behaviors, more agents remain on the team and thus generate more overall diversity concerns. That is, with transformational leadership enabled, the system sees less absences and nearly zero turnover. Therefore, the additional agents on staff over a longer period of time generate more total concerns for the leader to address.

Additionally, even though the numbers of diversity concerns are higher, the fact that the teams still achieve high values of internal network density is an indication that transformational leadership support behaviors are indeed working.

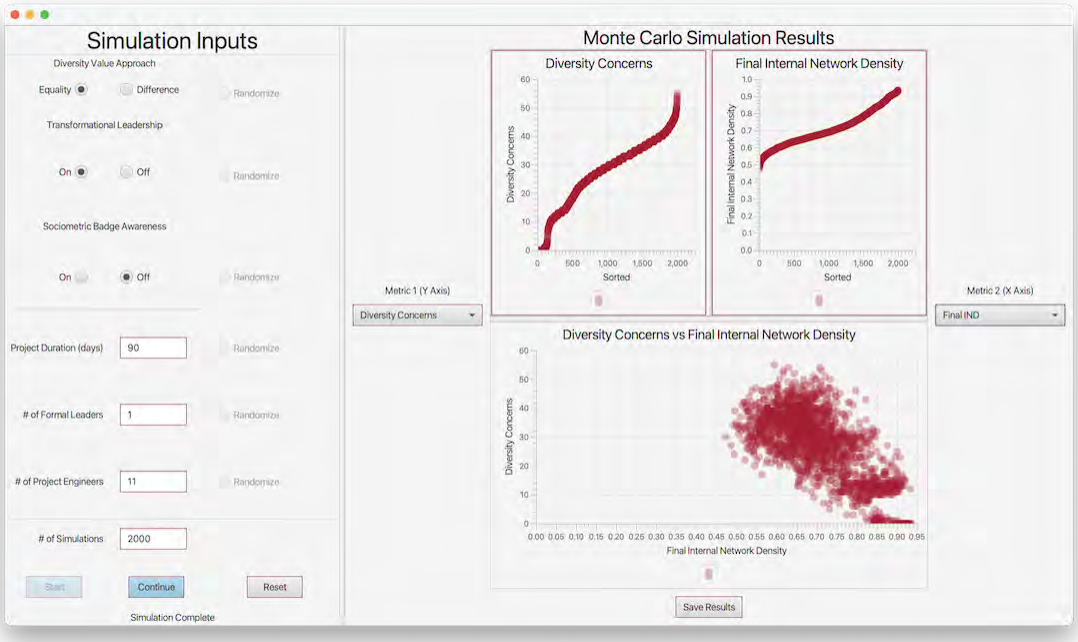

Figure 5-15: Transformational Leadership Monte Carlo Simulation Results: Diversity Concerns Vs. Internal Network Density *(Value in Equality)*

Figure 5-16: Transformational Leadership Monte Carlo Simulation Results: Diversity Concerns Vs. Internal Network Density *(Value in Difference)*

Correlation of Diversity Concerns & Agent Absences. The correlation of diversity concerns and agent absences is still positive, but as shown in Figure 5-17 and Figure 5-18, transformational leadership support actions eliminated the stair step data shapes seen in the base case scenarios (i.e., Figures 5-5 and 5-6). The leaders in this simulation identified and mitigated the concerns prior to the agents reaching their absence threshold.

Additionally, comparing the base case data with the current simulation indicates that transformational leaders cut the team absences down by approximately 50%. As noted earlier, additional empirical validation is needed to make this percentage and the absolute result values relevant, but the result indicates that significant drops in absences may be possible with successful transformational leadership behaviors. In the base case *value in equality* scenario, for example, the number of absences reached 30. With transformational leader support behaviors, however, the team with the most absences only reached 13. Similarly, the base case *value in difference* scenario teams topped out at 20 absences, whereas absences in transformational leader-led teams did not extend beyond 11. Though these absolute numbers are not necessarily significant to real-world teams due to the unvalidated assumptions in the model, they further illustrate that supportive behaviors have the potential to significantly reduce absences throughout a project.

Finally, by comparing the sorted values in each respective case, we can see that a significantly greater proportion of teams with transformational leaders had a small number of absences as compared to their counterpart base-case teams. This illus-

trates that although teams began to have absences, the supportive behaviors not only addressed the concerns, they also stunted any further absentee growth.

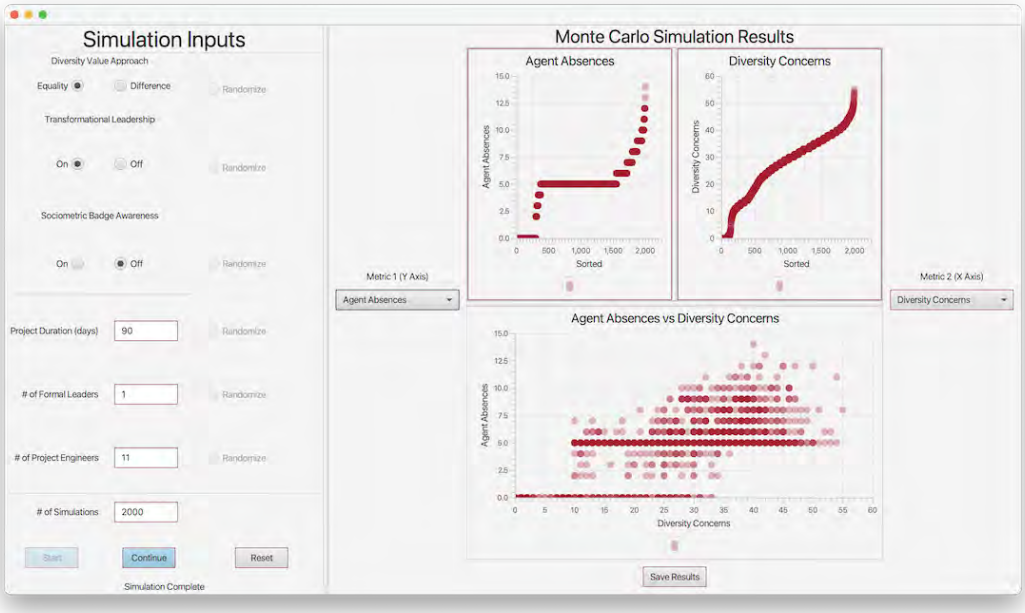

Figure 5-17: Transformational Leadership Monte Carlo Simulation Results: Absences vs. Diversity Concerns *(Value in Equality)*

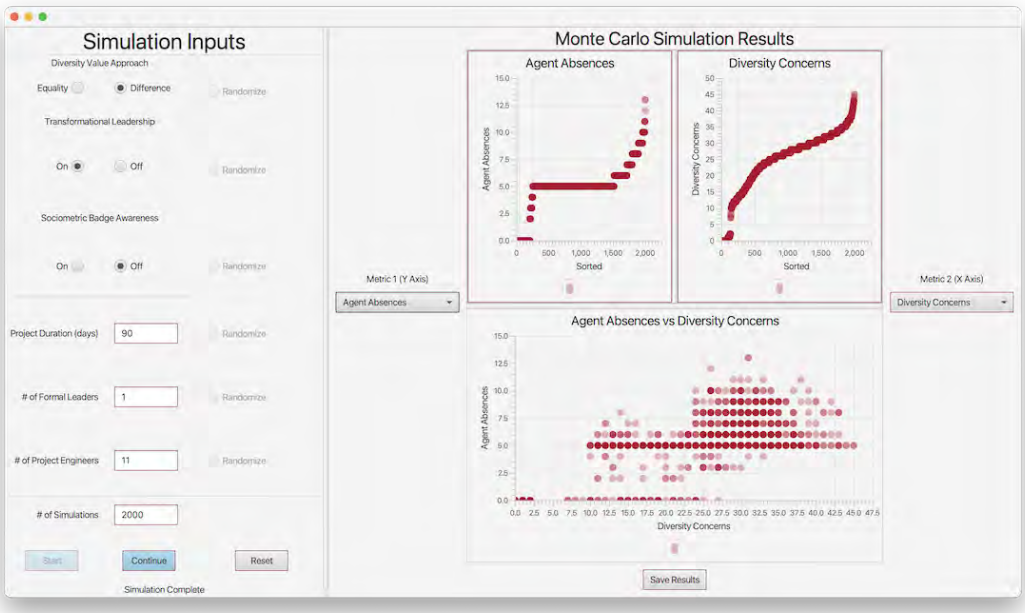

Figure 5-18: Transformational Leadership Monte Carlo Simulation Results: Absences vs. Diversity Concerns *(Value in Difference)*

Correlation of Leadership Interventions & Agent Absences. As might be expected, larger numbers of agent absences typically resulted in larger numbers of leadership interventions. As the model uses interventions to specifically mitigate the concerns of one struggling agent, this is an indication that more than one agent reached the intervention threshold in some of the simulation runs.

Specifically, in the majority of simulation runs, as Figures 5-19 and 5-20 show, one intervention was sufficient to arrest the negative concern-driven absence trend. However, in what appeared to be a very small number of cases across both scenarios, more than one agent on the team struggled to the point of needing an intervention. This phenomenon represents a realistic case of a single intervention not being sufficient for the entire team.

Along these lines, it is important to recall here that the model uses absences simply as a trigger to transformational leaders that the team is struggling. Though the model simulates leader agents using interventions to address concerns of specific agents, it also recognizes that diversity concerns can continue to grow even after such a collective team-building event takes place. Therefore, once a leader agent conducts such an intervention, the model simulates the leader using information gained from the event to address lingering concerns as well as concerns that continue to grow. These daily support behaviors are discussed in the next section.

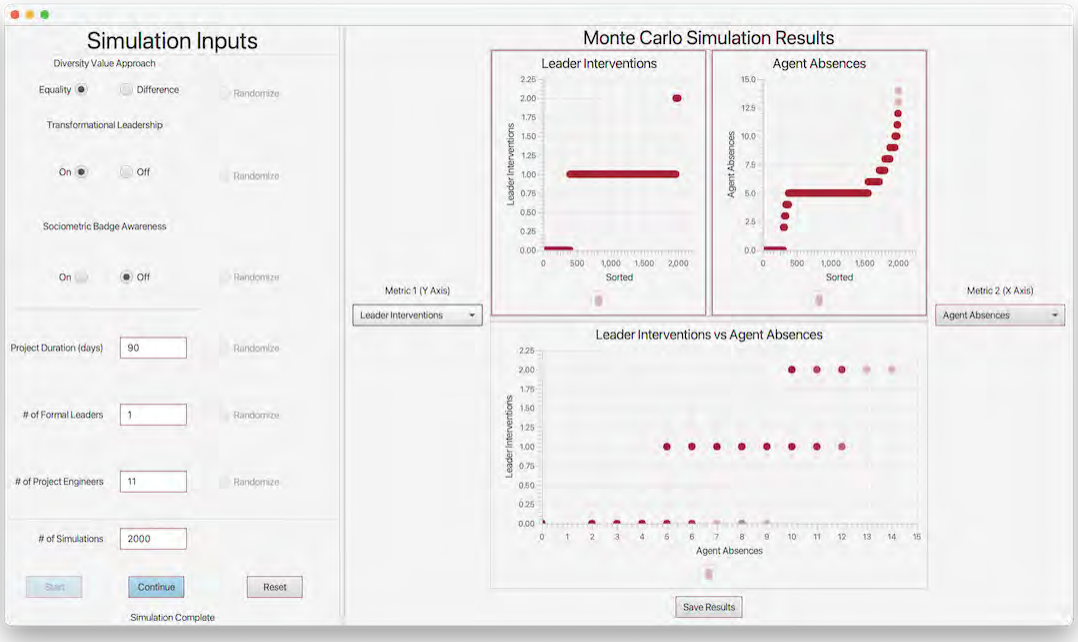

Figure 5-19: Transformational Leadership Monte Carlo Simulation Results: Interventions vs. Absence *(Value in Equality)*

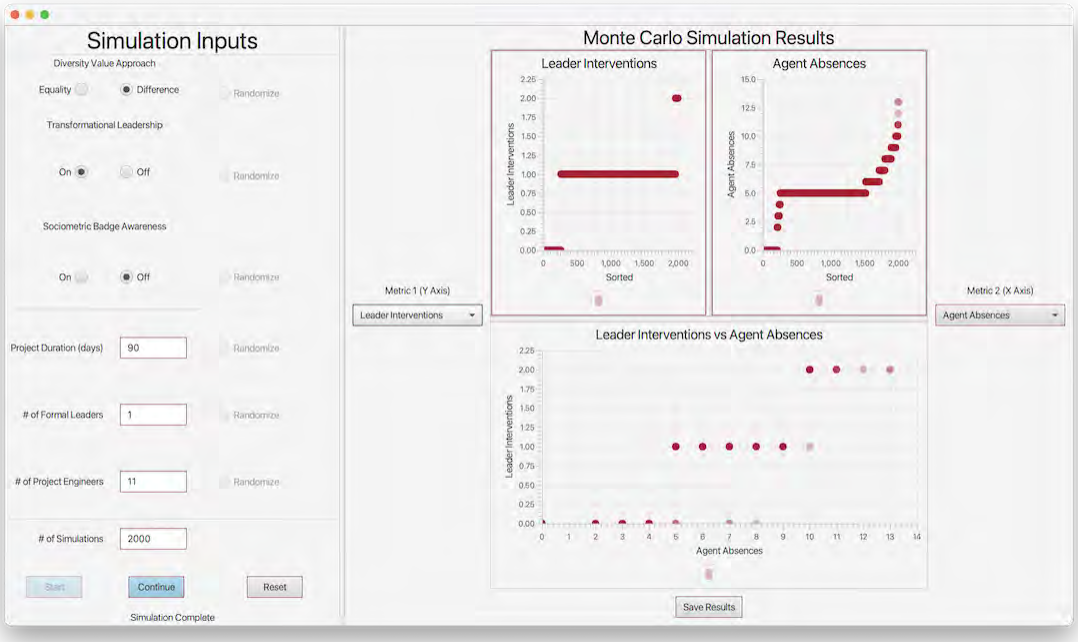

Figure 5-20: Transformational Leadership Monte Carlo Simulation Results: Interventions vs. Absence *(Value in Difference)*

Correlation of Leadership Daily Support & Diversity Concerns. The simulation results, as shown in Figures 5-21 and 5-22, show that as diversity concerns rise, so also do leader daily support behaviors. Interestingly, the results also clearly show that the leader agents are not addressing every concern that is generated, which likely would also be impossible in empirical studies. As previously noted, this is expected as the model allows leader agents to work on non-diversity-related tasks if the diversity concerns on the team are less than 25% of the related threshold.

Additionally, not satisfying every diversity-related concern, yet still maintaining the entire team's persistence throughout the project, is an important attribute in the model. Specifically, it captures the reality that leader agents can not be perfect, yet they can still achieve better results over the base case scenarios. As further research into these team dynamics occurs, one interesting question to address might be to investigate how much a leader tries but fails to address the concerns correlates with the team members' perceived support from the leader.

Along these lines, one final insight these Monte Carlo results offer is that it is unclear when the concerns are generated in comparison with when they are mitigated. That is, because this particular simulation needs the leader agents to recognize their team members are too absent before beginning to address some of these lingering concerns, and because the Monte Carlo simulation only captures final data values, the length of time the concerns have been lingering is unclear. For example, the lower number of leader daily support behaviors in relation to the concerns generated may be somewhat related to absences taking too long to accrue before the leader addresses

them, but this is speculation. More detailed data would be required to address this question, as will be seen in Section 5.3.

Figure 5-21: Transformational Leadership Monte Carlo Simulation Results: Support vs. Diversity Concerns *(Value in Equality)*

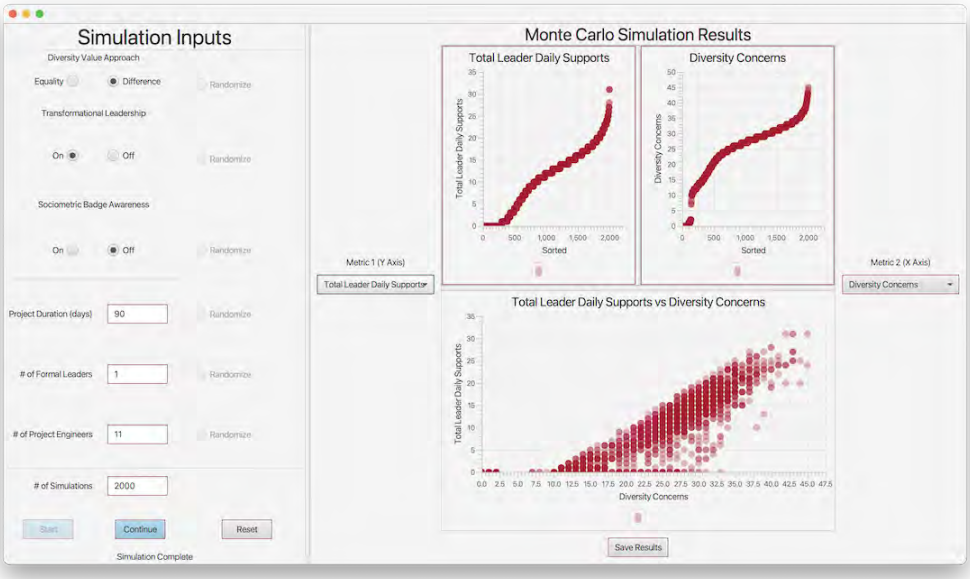

Figure 5-22: Transformational Leadership Monte Carlo Simulation Results: Support vs. Diversity Concerns *(Value in Difference)*

Correlation of Absences & Agent Departures. The simulation resulted in zero agent departures across all runs in both *value in equality* and *value in difference* scenarios. Therefore, the respective correlations are not applicable. However, this is one of the two significant results of the simulation indicating that transformational leadership support behaviors ultimately encouraged each of the member agents to persist with the project. Though not particularly insightful, Figure 5-23 illustrates this result.

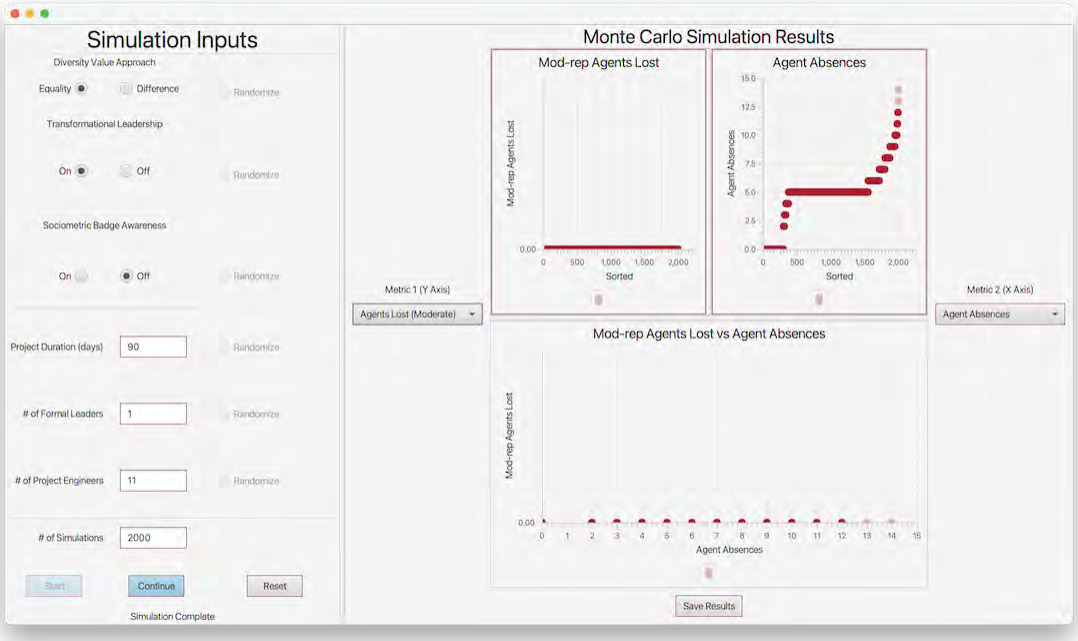

Figure 5-23: Transformational Leadership Monte Carlo Simulation Results: Agent Departures vs. Absences *(Value in Equality)*

Correlation of Agent Departures & Project Duration. In accordance with the simulation not resulting in any agent departures across scenarios testing either diversity approach, the agent departure correlations with project duration are also not applicable. However, this is the second of two significant simulation results, indicating that transformational leadership support behaviors can improve project performance by limiting any schedule delays resulting from diversity-concerns. The resulting plot from the *value in difference* simulation is shown in Figure 5-24.

The star in the lower left corner of the "Project Duration vs. Low-rep Agents lost" plot represents the Utopia point of this simulation. That is, the most-desirable is a minimal project duration with minimal agents lost.

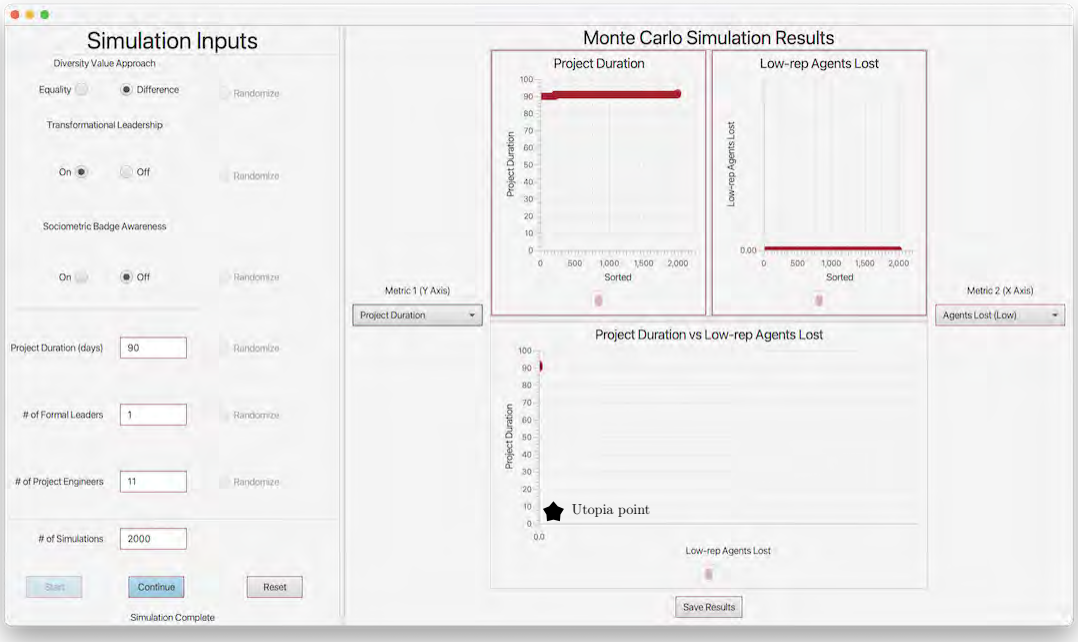

Figure 5-24: Transformational Leadership Monte Carlo Simulation Results: Project Duration vs. Agent Turnover *(Value in Difference)*

5.2.2 Single Simulation Results

Next, just as seen in the base case scenarios, single simulations also provide additional insights into the teams with transformational leaders as well. Similar to Section 5.1.2, this section discusses results of sample single simulation runs for each diversity approach.

***Value in Equality* Transformational Leadership Scenario.** Agents on the team shown in Figure 5-25 accrued diversity-related concerns significantly faster than the sample team shown in Figure 5-11. Accordingly, they also started accruing absences faster as well. Additionally, the results show, as intentionally modeled, that prior to these absences starting to rise, leader agents did not take notice of the concerns on the team. Once they notice the absences rising and conduct the intervention, however, their resultant increased awareness allows them to use time each day to address any lingering team member concerns.

Moreover, the results also show that this team's average internal network density started falling at the start of the project. Though the initial trend for agents in the low-representation social group was positive, it was negative for all other social groups. Therefore, early interactions between each social group had an overall negative effect on the team that is not otherwise seen in the Monte Carlo simulations.

Interestingly, however, once the leader agent conducts an intervention and begins addressing the team's otherwise unseen concerns, the internal network density across

all social groups begins to rise. This is an emergent quality of the model that occurs as a result of leader agents addressing concerns from both agents involved in an interaction (see the drop in the average internal network density of the moderately-represented social group in Figure 5-26 for a counter example). That is, as the leader continues to address concerns throughout the team, all ties get stronger.

Finally, as a result of the leader agent minimizing the number of absences to , the agents on this particular team were able to complete the project with only a one-day delay.

Figure 5-25: Transformational Leadership Scenario Single Simulation *(Value in Equality*

***Value in Difference* Transformational Leadership Scenario.** As Figure 5-26 illustrates, members of this particular team reached significantly lower internal network densities before beginning to stay home from work. The "Diversity Consequences" chart further reveals that the leader was able to address two-thirds of the concerns with the intervention, indicating that one member in particular accrued the majority of concerns. The chart goes on to show that as the leader addressed the lingering concerns, the team did accrue additional concerns, but at a much slower rate that the leader was able to manage.

As a result of the leader addressing the open concerns, the average internal network density of the low-representation social group recovered. However, that of members in the moderately-represented social group began to drop late in the project. This is indicative of the leader taking time to address lingering concerns prior to addressing new ones. Once addressed, however, the social group's ties also recovered. These daily

internal network density dynamics are not captured in the Monte Carlo simulations, thus illustrating the benefit of capturing daily dynamic data.

Figure 5-26: Transformational Leadership Scenario Single Simulation *(Value in Difference)*

5.3 Sociometric Badge Experiment Simulation Results

Building on the previous base case and transformational leadership simulation results, this section simulates transformational team leaders with access to insights from team members' sociometric badge data. Importantly, this thesis assumes that researchers with access to the raw sociometric badge data will be able to correlate it with diversity concern and/or internal network density survey data as described in Section 3.3. The team leaders then work with the researchers to determine if there are concerns on the team, and then work with the team to address the concerns.

As in the previous scenarios, this section first discusses results and insights from two Monte Carlo simulation results that vary with diversity approach. It then concludes with insights from two corresponding sample single simulation results.

5.3.1 Monte Carlo Results

The Monte Carlo simulations of leaders having access to sociometric badge data yielded similar results to those of transformational leaders without it, with a few

notable exceptions. For example, because leaders in both scenarios needed some type of trigger (i.e., absences and researcher-correlated insights), lead time allowed the team to generate diversity concerns. Additionally, leaders in both scenarios were able to encourage agents to persist throughout the entire project, which in turn led to very small delays. Notably, however, access to sociometric badge data gave leaders more lead time to prevent any agent absences, which in turn led to zero delays across both of the entire 2000-run simulations.

The correlation results from these *value in equality* and *value in difference* simulation data are presented in Table 5.4 and Table 5.5, respectively. Because there were no absences, agents lost, or project delays in the simulations, the matrices do not include these variables.

	1	2	3	4
1. Interactions	1			
2. Internal Net Density	-0.043	1		
3. Diversity Concerns	-0.020	-0.70	1	
4. Daily Supports	0.019	-0.63	0.99	1

Table 5.4: Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Equality)*

	1	2	3	4
1. Interactions	1			
2. Internal Net Density	0.030	1		
3. Diversity Concerns	-0.052	-0.68	1	
4. Daily Supports	-0.052	-0.68	0.99	1

Table 5.5: Correlation Matrix: Transformational Leadership Monte Carlo Simulation *(Value in Difference)*

Correlation of Interactions & Internal Network Density. Similar to the previous scenarios, the results continue to indicate no significant correlation between internal network density and the number of interactions the team members have. They also, however, continue to provide noteworthy insights.

For example, the teams with the lowest internal network densities led by leaders with access to sociometric badge data still had ties that were stronger than the previous scenarios (see Figure 5-27 and Figure 5-28). As the figures show, both simulations resulted in a minimum value of not less than 0.6, whereas all previous simulations had small numbers of runs that dropped between 0.4 and 0.5.

Moreover, as seen in the "Final Internal Network Density" plots in the upper left corner of each figure, approximately 75% of the results are above 0.75, whereas only about 25% and 50% of the base case and transformational leadership cases, respectively, were about 0.75. This plot also shows that approximately 5% of the values fell below 0.75, approximately 10% of the values were above 0.80, and the rest were uniformly distributed between 0.75 and 0.8. This is in stark contrast to the previous simulations which resulted in more continuous distribution between their respective

minimum and maximum values. In other words, team leaders with sociometric badge-like insights built stronger teams, and they did it more predictably.

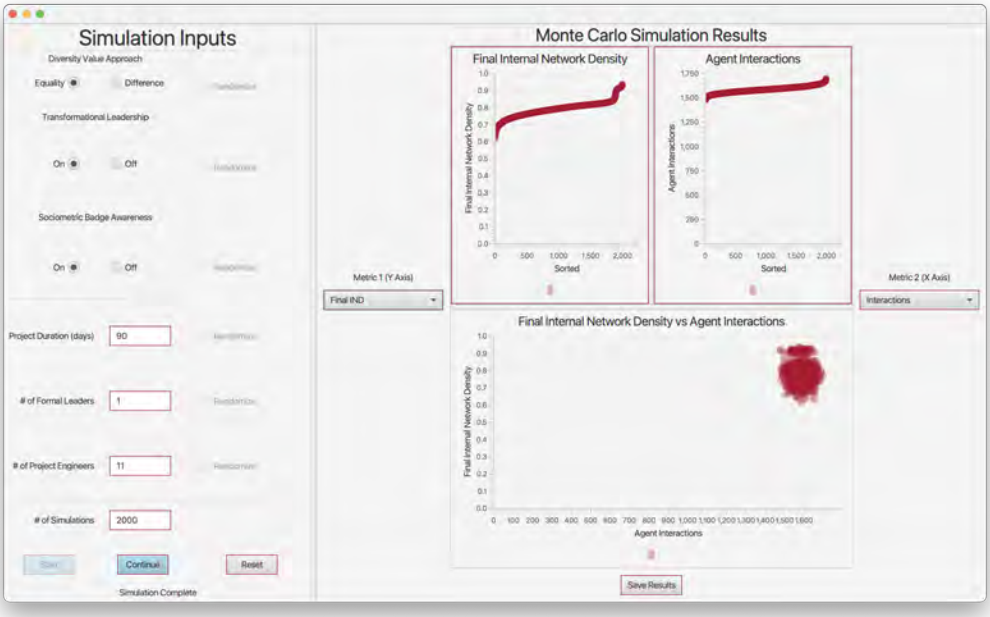

Figure 5-27: Sociometric Badge Awareness Monte Carlo Results: Internal Network Density vs. Interactions *(Value in Equality)*

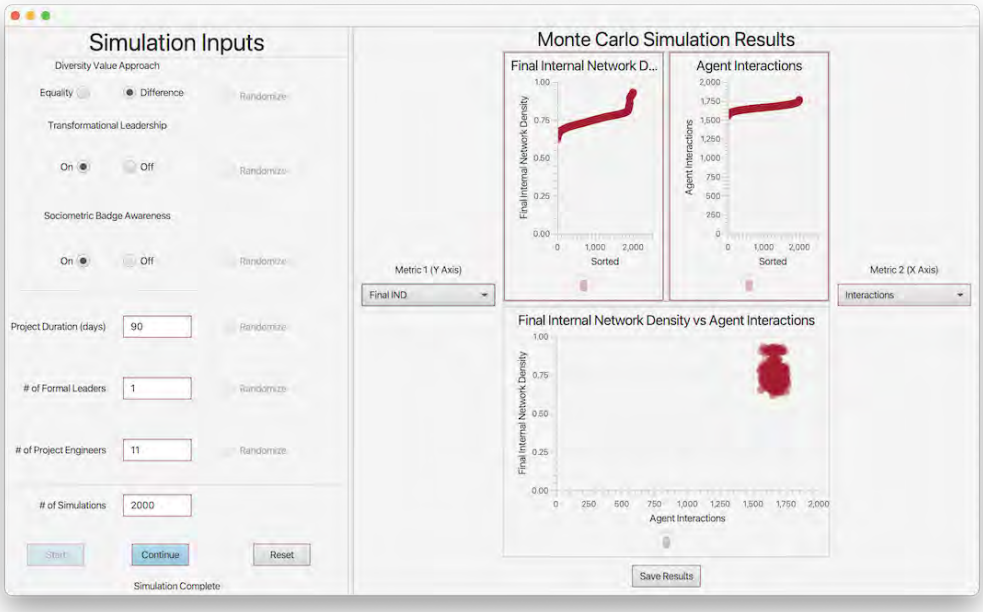

Figure 5-28: Sociometric Badge Awareness Monte Carlo Results: Internal Network Density vs. Interactions *(Value in Difference)*

Correlation of Internal Network Density and Diversity Concerns. As in all other scenarios, the number of diversity concerns the team generated is strongly and negatively correlated with the team's final internal network density (see Figure 5-29 and Figure 5-30). Additionally, the teams in this scenario all resulted in relatively higher numbers of diversity concerns as well as higher final internal network densities. This phenomenon is amplified over the transformational leadership scenarios because leaders began addressing concerns earlier in each project run, thereby eliminating agent absences and again generating more concerns overall.

However, an important insight emerged from the teams with sociometric badge awareness. Both of the previous base case and transformational leadership scenarios resulted in more runs that ended with internal network densities above 0.85 than the present scenario testing sociometric badge awareness. That is, even though the leaders with this additional awareness more consistently built stronger teams, it was the transformational leader scenarios that had the highest number of runs with these high density values (by a factor of two over the sociometric badge teams). This phenomenon results from the modeled benefit of successful interventions giving everyone on the team an internal network density boost, whereas the daily leader supports only affected the agents with diversity concerns. Further, it offers an important reminder that leaders may need to work with all team members rather than just the ones expressing concerns if they want to build the strongest teams.

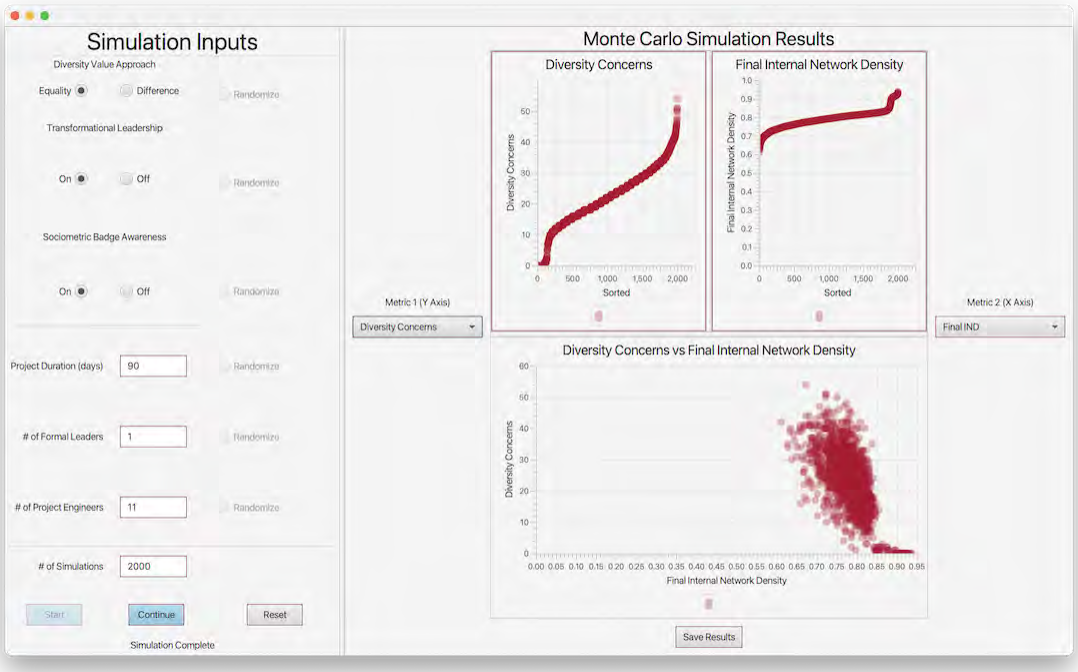

Figure 5-29: Sociometric Badge Awareness Monte Carlo Results: Diversity Concerns vs. Internal Network Density *(Value in Equality)*

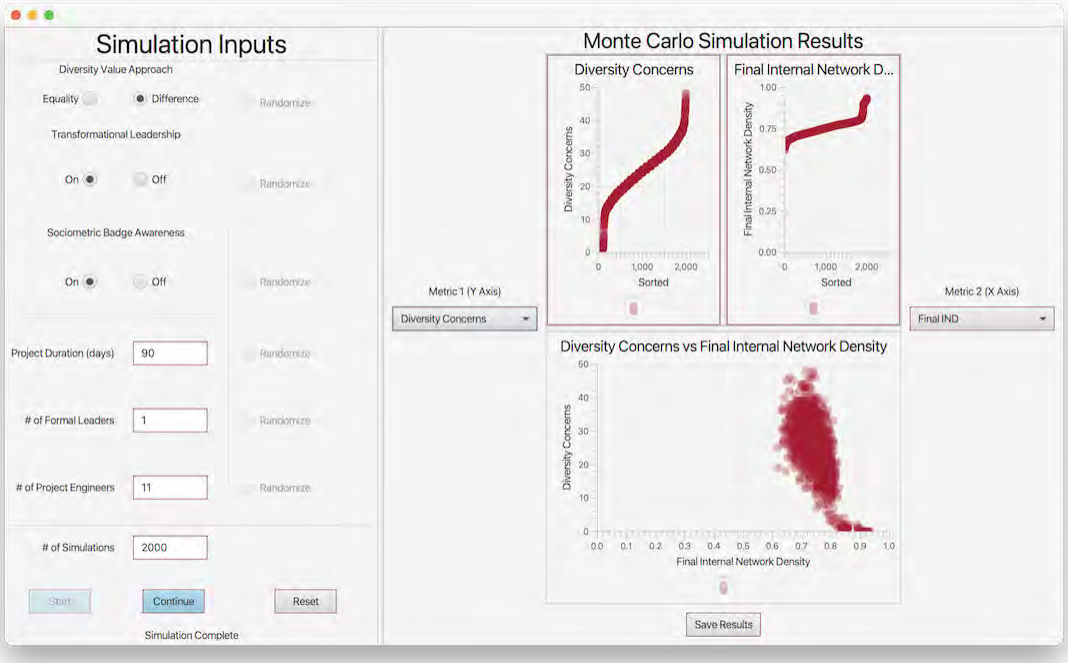

Figure 5-30: Sociometric Badge Awareness Monte Carlo Results: Diversity Concerns vs. Internal Network Density *(Value in Difference)*

Correlation of Daily Supports and Diversity Concerns. Like the previous scenarios testing transformational leadership behaviors, adding in sociometric badge awareness also showed a strong positive correlation between the number of generated diversity concerns and the number of daily leader support actions. Indeed, the relationship was even stronger with sociometric badge awareness, and as seen in Figure 5-31, these leaders indeed addressed more (but still not all) of the concerns on the team [Note - the simulations resulted in both diversity approaches producing similar phenomena; therefore, the *value in difference* plot is omitted for clarity].

Correlation of Agent Absences and Project Duration. Every leader in the simulation with sociometric badge awareness successfully prevented any absences from occurring on the team. As a result, there were also no associated project delays. Therefore, such a correlation is not applicable. However, though such a chart is not particularly insightful, Figure 5-32 presents an important result of this thesis. That is, daily support behaviors were shown to eliminate agent absences, as well as their associated departures and project delays [Note - the simulations resulted in both diversity approaches producing similar phenomena; therefore, the *value in equality* plot is omitted for clarity].

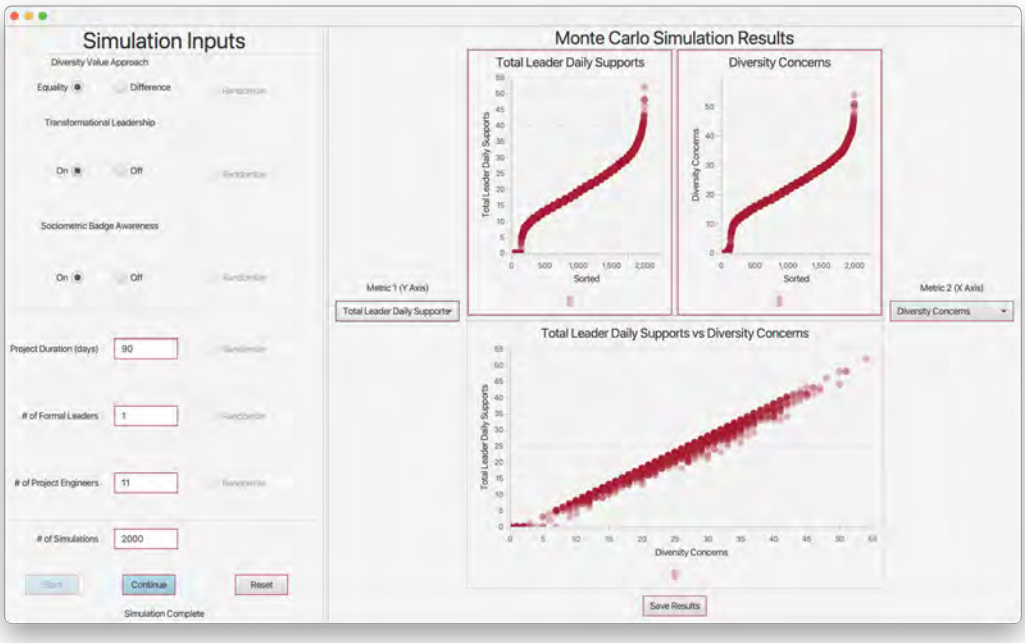

Figure 5-31: Sociometric Badge Awareness Monte Carlo Results: Daily Leader Supports vs. Diversity Concerns *(Value in Equality)*

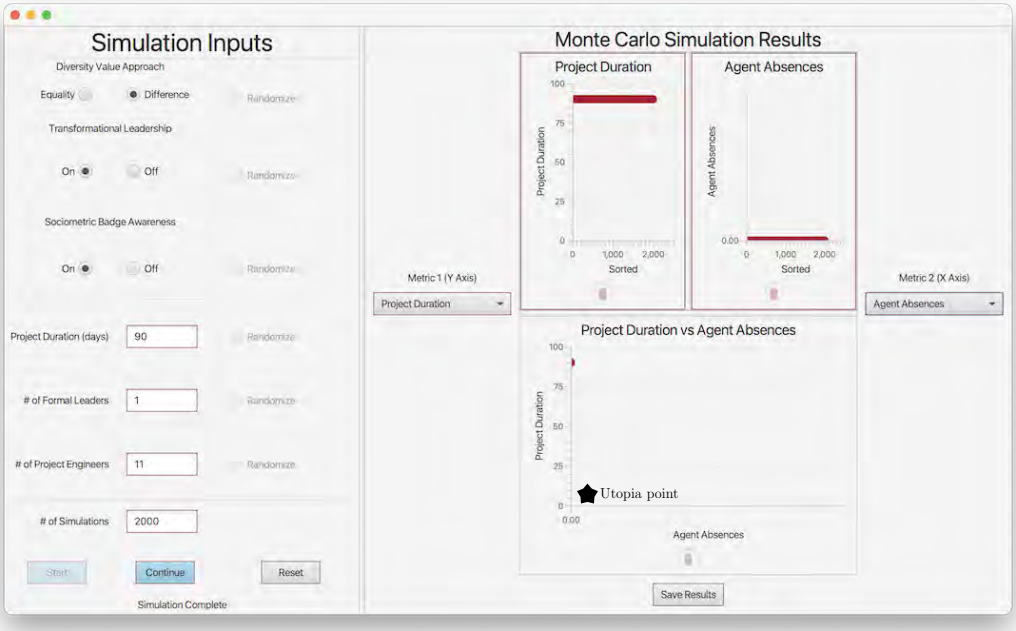

Figure 5-32: Sociometric Badge Awareness Monte Carlo Results: Project Duration vs. Absences *(Value in Difference)*

5.3.2 Single Simulation Results

As in the previous scenarios, reviewing sample single simulation runs provides further insights not observable in the Monte Carlo simulations. This section discusses two such sample simulations.

***Value in Equality* Sociometric Badge Awareness Scenario.** In this scenario, the high- and moderately-represented social groups started the project with higher average internal network densities than their low-represented peers. With a *value in equality* diversity approach, the low-represented ties began to grow stronger, whereas those of the moderately-represented social group initially weakened as the interactions occurred.

After a brief period simulating time for researchers to correlate sociometric badge data with diversity concern and internal network density survey data, the leader begins to work with individual team members to address concerns. However, as indicated by the diversity concern line not going to zero, some concerns linger. Therefore, though the leader was able to maintain a strong team internal network density, its choices to focus on other work left concerns that ultimately started to bring that average down as the project neared its completion.

Figure 5-33: Sociometric Badge Awareness Experiment Single Simulation *(Value in Equality)*

***Value in Difference* Sociometric Badge Awareness Scenario.** This scenario dramatically amplifies the impact of lingering concerns seen in the previous sample run. Unlike the previous team, this team started the project with both the low- and moderately-represented social groups exhibiting weaker internal network densities than their highly-represented peers. As the agents interacted with each other, the *value in difference* approach enabled the tie strength of the moderately-represented group to grow.

However, that of the low-represented group suffered considerably, even though the leader was addressing diversity-related concerns. This phenomenon occurs for two reasons. First, there are less agents in the low-represented group, and therefore, average values are more easily impacted. More importantly, however, this results because the lingering concerns not addressed by the leader reside with individuals with which the low-represented agents interacted. This in turn prevented the low-represented agents from building strong ties with the other agents. That is, though the leader was addressing some of the concerns, not supporting every team member had a particularly strong impact on the low-represented group, which in turn brought down the overall team's average internal network density. Around day 70, however, the leader addressed some of these other concerns, which in turn had a dramatic effect on the low-represented social group's trending strength.

Figure 5-34: Sociometric Badge Awareness Experiment Single Simulation *(Value in Difference)*

5.4 Significant Result: A Goal to Strive For?

In addition to the insights offered by the visualizations developed as part of this effort's agent-based simulation, we can gather further insights by using commercial software to visualize the data as well.

For example, another way to look at the data shown in Figure 5-21 (transformational leadership simulation of a *value in equality* approach without the benefit of sociometric badge awareness) is to categorize the points according to the project's total number of absences. Recalling that this scenario resulted in 100% agent retention, the new plot shown in Figure 5-35 reveals that although transformational leaders may not have to address all concerns and still be able to retain their personnel, such a strategy has increased risk of higher absences. Again, though the absolute numbers need to be validated against empirical data, this graph illustrates that there may be a "leader effectiveness" (i.e., concerns satisfied in relation to concerns generated) goal to strive for in association with an acceptable absence rate. Such an insight could help drive further transformational leadership training programs.

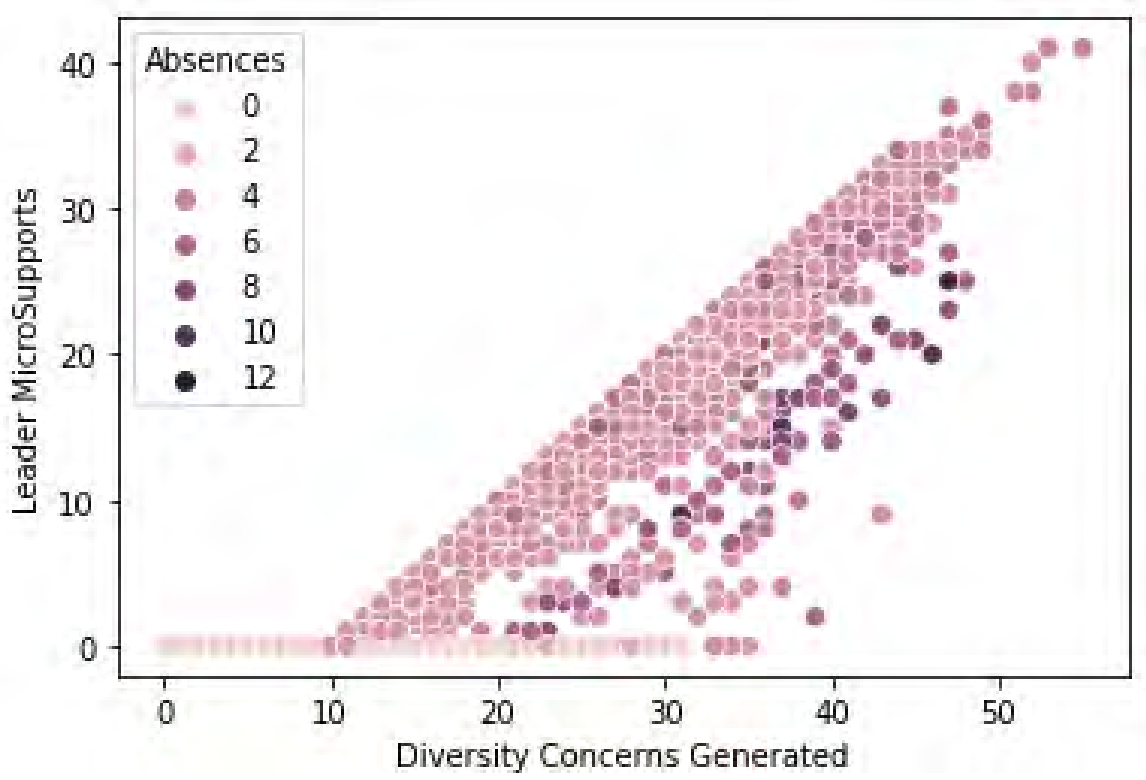

Figure 5-35: Caption

Further, as Table 5.3 reports, the number of absences is positively correlated with the diversity concerns that the team generates. By categorizing original Figure 5-17 according to such a "leader effectiveness" calculation mentioned above, we get a new insight into the challenges these leaders face (see Figure 5-36). That is, as the number of absences on the team and the number of diversity concerns they generated grew,

the leader agents were less likely to achieve high levels of effectiveness. This is due primarily to the code's 2-hour/day limit for leaders to address concerns, and though 2 hours per day was sufficient for this simulation, it may not be true for longer projects or bigger teams. This highlights a need to better understand leader effectiveness required with different team sizes or project spans.

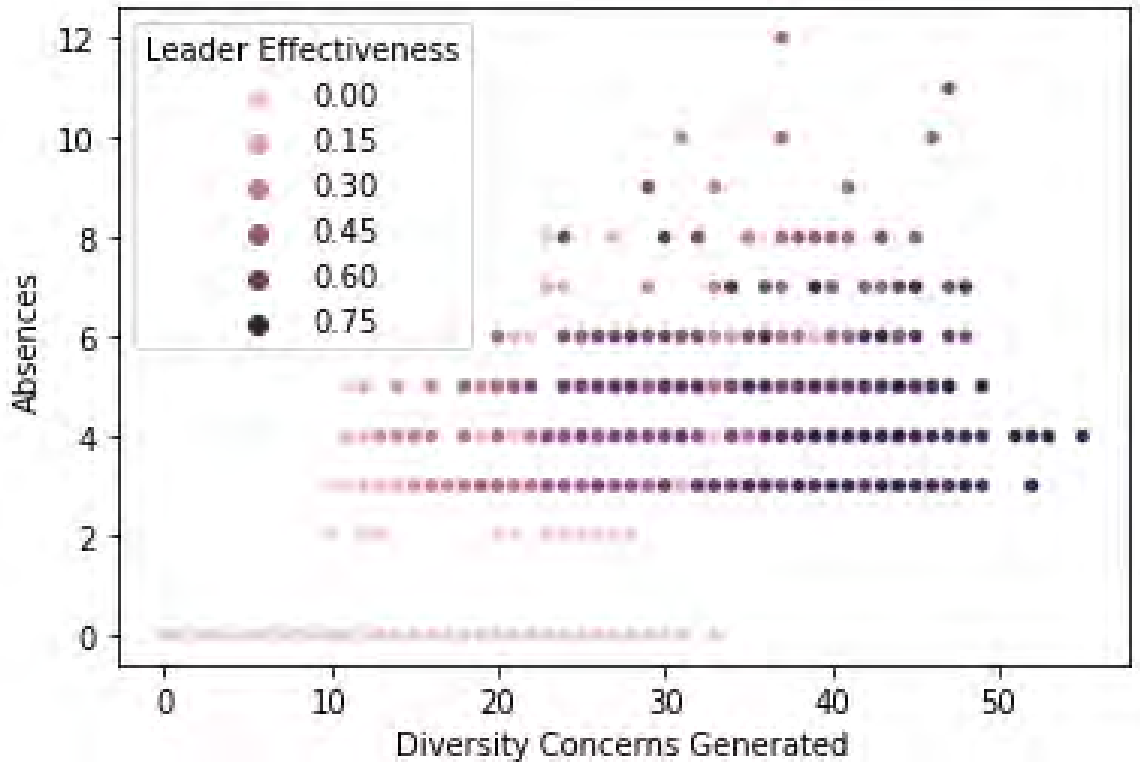

Figure 5-36: Transformational Leadership Scenario: Absence vs. Internal Network Density & Leader Effectiveness *(Value in Equality)*

5.5 Chapter Summary

This chapter presents the results of three scenarios, each using Monte Carlo simulations as well as representative sample single simulations to present important insights for leaders to consider. These simulations together suggest that leaders with access to both intermediate, dynamic data from team member engagements, as well as predicted results from systemic models like that pictured in Figure 3-4, may achieve improved team performance as a result of earlier and better leadership decisions.

The first scenario, referred to as the base case, used the model developed in this thesis to simulate a diverse team of 11 engineers working together on a 90-day project. Next, the model was used to simulate transformational leaders using agent/team member absences as a trigger to employ transformational leadership support behaviors. Finally, the model simulated transformational leaders working with researchers using sociometric badges to track and correlate team member interactions

with diversity concerns.

Results from these scenarios were compared and contrasted throughout the chapter. In general, the base case scenario illustrated agent absences and departures in accordance with the diversity-concern patterns identified by Apfelbaum et al. (2016). Further, once the transformational leader agents recognized these absences, they were able to address the team, mitigate lingering concerns, and address future concerns in a way that minimized further absences and avoided any departures. Finally, the teams with leaders that had access to sociometric badge data were able to consistently build strong teams, but not always the strongest due to their leader's focus on just agents that had concerns.

While the majority of the results themselves are more reflective of past research, they are important to the foundation for further model development. That is, the results presented herein indicate that the model has potential to develop even further insights, especially as further dynamic research is collected and incorporated into the code.

The next chapter concludes this thesis with a concise consolidation of the significant insights identified in this chapter, as well as a discussion of this effort's limitations. It also offers some thoughts for consideration of future research in this area.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Conclusions, Limitations, and Future Research Suggestions

> "So, let us not be blind to our differences–but let us also direct attention to our common interests and to the means by which those differences can be resolved. And if we cannot end now our differences, at least we can help make the world safe for diversity. For, in the final analysis, our most basic common link is that we all inhabit this small planet. We all breathe the same air. We all cherish our children's future. And we are all mortal." - John F. Kennedy, American University Commencement Address, 10 June 1963

This thesis collected research associated with teams, diversity, and leadership to identify potential solutions to the diversity-related performance and persistence gaps highlighted by Apfelbaum et al. (2016). Namely, with the insights their study uncovered regarding how an organization's diversity approach might affect retention of team members according to their social group's representation within a larger group, my research effort set out to find a way to increase retention of all team members regardless of social group. Specifically, after conducting the review of available literature, I sought answers to the following research question:

> ***Research Question -*** *How might transformational leadership behaviors associated with support affect a diverse team's internal network density and associated performance and organizational retention?*

The remainder of this chapter summarizes my conclusions, the limitations of my research, and offers suggestions for future work.

6.1 Conclusions

6.1.1 Summary

As a System Design & Management student, I started my journey reviewing an extensive body of research associated with the systemic nature of teams. After a look

through history, I found that today's research is generally focused on the dynamic nature of teams as opposed to some of the older Input-Process-Output studies. Further, this review revealed that today's technology, like sociometric badges for example, can be helpful in relating daily, micro-level interactions to macro-level organizational results.

Next, my survey of the associated literature then revealed another extensive body of research on the topic of diversity, but unfortunately, many of the studies presented results that conflicted with those of other studies. Indeed, I found that scholars tend to refer to diverse teams as "double-edged swords" (Williams and O'Reilly III, 1998) because although diversity has been shown to add creative benefits and increased performance over homogeneous teams, most empirical studies found diversity to be a negative contributor to performance.

However, Reagans et al. (2004) offered a potential rationale for these conflicting results: diverse teams typically have weaker internal network densities, but stronger external network ranges. Since both structural network variables have been shown to be positively correlated to team performance, it stands to reason that the internal network density structural variable on a team would represent a significant factor in whether or not a team performed well. Therefore, improving the magnitude of a team's weaker internal network density represented a key area team leaders could focus on for better overall results.

Next, to address how leaders might improve internal network density, and recognizing that creativity represents a common benefit of diverse teams, I reviewed leadership studies on creativity and found that perceived leader support (Amabile et al., 2004) is a key contributor to positive team performance. Of 14 commonly-accepted leadership behaviors (Yukl and Van Fleet, 1992), Amabile et al. (2004) showed 6 of them to have a significant impact on perceived leader support (i.e., Monitoring, Recognizing, Supporting, Rewarding, Problem solving, and Motivating & inspiring).

The creativity-focused review also revealed that a leadership style known as transformational leadership (Burns, 1978) is positively correlated with team creativity and performance. These leaders are known to enhance motivation, morale, and the morals of their teams by building on their values and changing their mindsets such that they strive for higher-level goals (Amabile, 1983; Tierney et al., 1999; Jung, 2001). Further, Tracey and Hinkin (1998) found that of the 14 Yukl and Van Fleet (1992) leader behaviors, 4 of them are most closely associated with transformational leaders (i.e., Motivating & inspiring, Supporting, Team Building, and Clarifying). Cross-referencing this set with the 6 behaviors associated with perceived leader support, two transformational leader behaviors stood out as potential areas to study: Motivating & inspiring, and Supporting.

While Amabile et al. (2004) showed Motivating & inspiring behaviors to be significantly correlated with team performance, the polarity of the significant behaviors was negative. That is, leaders that tried to motivate and inspire but failed (or leaders that didn't try to motivate and inspire at all) had a much larger negative effect on their teams as compared to the positive effect that good motivators had on their respective teams. Contrary to this, Amabile et al. (2004) also showed that leaders who performed supporting behaviors well had a much more positive effect on their

teams as compared to the smaller negative effect that poorly-supportive leaders had on theirs.

For this reason, as revealed in the research question, I elected to focus my study on how supportive behaviors might affect a diverse team's internal network density. In particular, these types of behaviors include: "Acting friendly and considerate, being patient and helpful, and showing sympathy and support when someone is upset or anxious, listening to complaints and problems, and looking out for someone's interests (Yukl and Van Fleet, 1992)." Or perhaps more simply, they mean knowing when to be a good friend.

6.1.2 Experiment Insights & Conclusions

Combining all of this research together, I designed a team-based experiment using sociometric badges to collect and correlate interactive data with reported concerns and internal network densities from team members. Unfortunately, due to Corona Virus-19 restrictions and my own limited fellowship availability, I was not able to conduct such an experiment. Instead, I leveraged the results of my research review to construct an agent-based model that not only might offer some initial insights, but might also serve as a first step for follow-on research.

After validating the mechanics of my agent-based model (ABM), I simulated the experiment I designed in a number of 2000-run Monte Carlo simulations. I also used a single-simulation functionality to identify more daily dynamic insights that would not otherwise be seen in a Monte Carlo fashion.

In general, though the initial version of the code is limited, a number of insights came out of this work. First, when leader agents were supporting each of their team members, the interactions that the members had with each other ultimately built stronger overall ties. On the other hand, if leaders were only supporting one agent in a pair of interacting agents, the interaction could still lead to a more negative tie strength and overall internal network density. This insight highlights the importance of understanding and addressing all team member concerns rather than focusing on those that might be most obvious.

Another insight is based on the fact that leaders have limits to the level of support they can provide their teams. I assumed a limit of 2 hours/day (approximately 25% of work time), which was sufficient to retain all agents on the teams my model generated. However, this might not be the case for larger teams, suggesting that informal leaders on the team might need to help as well (Morgeson et al., 2010). Therefore, as organizations consider how to train their formal leaders how to be more supportive of the teams they lead, the organizations should also consider including their more junior members in this training as well. The junior members can then serve as informal leaders to help the leader spot and address these types of concerns.

Finally, continuing the thread of leader limitations, one last insight hinted at in this work is that the amount of coordination could be a significant contributor to leader performance. As mentioned in Section 2.2.2, task interdependence has been shown to positively moderate diverse team performance, but not enough to overcome the negative aspects of diversity (Horwitz and Horwitz, 2007). With a hard-coded

coordination factor of 20% and the fact that not all interactions led to diversity concerns, the hard-coded leadership time budget of 25% mentioned previously gave the leader agents sufficient resources to address enough team concerns to mitigate agent turnover. As teams grow and leader responsibility increases, this coordination factor will be an important area to watch.

6.2 Limitations

The empirically-unvalidated nature of the agent-based model represents a significant limitation of this work. While the mechanics of the model are based on peer-reviewed research, the actual values used in the calculations need significant work. For this reason, none of the actual values resulting from this effort should be taken at face value. Instead, only trends and comparisons should be drawn at this time.

Additionally, some of the initial assumptions I made to get started introduce significant limitations as well. For example, the model assumes that the leader agents are trained in transformational leadership support behaviors, and that when they elect to use them, they are successful. There is no accounting for weaker leaders at this time.

Another similar limitation is that there are no adverse affects of agents taking the day off other than delayed project completion. In fact, the model currently simulates that any agent can pick up where another agent left off to complete the task. This is a result of coding the number of tasks according to the project span and the number of agents on the team. It is not a completely realistic case, and it limits insight to the scope of potential delays associated with sharing tasks.

Along these lines, the code also does not currently include any efficiency factors associated with interactions themselves. Instead, it assumes that an agent pair work flawlessly together to complete the task. As an update to the model, it may be appropriate to consider the agent pair's tie strength as a factor when determining how efficient they are.

Finally, while there are likely significantly more limitations associated with this work, one final one worth noting here is that the code currently assumes that homophily drives stronger network ties among agents. However, this is likely not always the case. After all, even among homogeneous teams, there still might be competition or other variables that limit the tie strength among team member pairs. The code does not include any logic along these lines.

6.3 Suggestions for Future Research

My strongest recommendation for future research is to conduct an empirical experiment in accordance with the design presented in Section 3.3. In accordance with valuable insights gained even from an unvalidated agent-based model, such an experiment could help drive leadership development programs towards more effectively supporting team members, and thus achieving better team results.

Additionally, as alluded to in Section 6.1.2, transformational leader effectiveness with respect to supporting team members represents a key area of future research. Questions associated with how to balance these supportive actions with other tasks such that leaders do not fall behind should be addressed. For example, though the other 13 Yukl and Van Fleet (1992) behaviors may not have as significant an affect on overall team performance, many of them are still needed for teams to function. Therefore, questions associated with balancing these leadership-related tasks specifically as they relate to diverse teams in dynamic settings might also be addressed.

Similarly, as interactions and internal network density drive the generation of diversity-related concerns, it stands to reason that there may be limit to the amount of coordination in a diverse team project that one leader should be expected to manage without help. This is not to say that diverse teams should pose an additional leadership burden by adding more leaders to the team, but that members of the team (in addition to the leader) should be shown how their supportive behaviors can also help address concerns and lead to more positive performance by everyone on the team (Morgeson et al., 2010).

Finally, as Katherine Phillips is quoted at the end of Chapter 2, "Diversity is hard." This thesis makes no attempt to address the potential toll on leaders as they work to mitigate growing numbers of diversity-related concerns. For example, it is unclear if a leader knowing that he or she is retaining their team members as a result of increased perceptions of leader support is enough to keep the leader going, or if the strain has negative implications that reduce the leader's performance. If the latter, what are the best ways to support the leader such that these implications are mitigated and everyone can perform at their best?

6.4 Closing Remarks

Representation among teams is a structural variable of the team. It can apply to race, gender, age, tenure, education or any other characteristic we might use to distinguish ourselves from each other via in- and out-groups. As such, leaders of any team are likely to have these diversity-related concerns present, but they may not know it if they are not properly trained to spot them. Further, if they cannot spot and address them, then they face greater risk of the negative effects that diversity can have on their teams. As leaders, it is not our place to ask for a business case associated with diverse teams. We must appreciate that diversity "is", and we must know how to lead our diverse teams through the related challenges. Only then can we be leaders known for inclusively and powerfully wielding our "double-edged swords" and attaining the bonuses that Scott Page writes about. I, for one, am in, and I hope you will join me. Thank you for reading.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A

Software Code

This appendix provides the code as of the date of this thesis. I am happy to provide the files upon request. Please feel free to contact me at jeremy.goodwin@sloan.mit.edu.

A.1 User Interface Controller Files

A.1.1 UserInterface.java

```
1
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.SplitPane;
5  import javafx.stage.Stage;
6
7  public class UserInterface extends Application {
8
9   public static void main(String[] args) {
10   launch(args);
11  }
12
13  @Override
14  public void start(Stage primaryStage) throws Exception {
15
16   var session = new DiversityModelRunner();
17
18   UserInterfaceInputView iv = new UserInterfaceInputView();
19   UserInterfaceResultsView rv = new UserInterfaceResultsView();
20
21   iv.session.set(session);
22   rv.session.set(session);
23
24   iv.sessionProperty().bindBidirectional(rv.sessionProperty());
25
26   SplitPane sp = new SplitPane();
27
28   sp.getItems().addAll(iv, rv);
29
```

```
30   Scene scene = new Scene(sp);
31   primaryStage.setScene(scene);
32   primaryStage.show();
33
34  }
35 }
```

A.1.2 UserInterfaceInputView.java

```
1
2 import java.io.IOException;
3
4 import javafx.beans.property.ObjectProperty;
5 import javafx.beans.property.SimpleObjectProperty;
6 import javafx.fxml.FXML;
7 import javafx.fxml.FXMLLoader;
8 import javafx.scene.control.Button;
9 import javafx.scene.control.Label;
10 import javafx.scene.control.ProgressBar;
11 import javafx.scene.control.RadioButton;
12 import javafx.scene.control.TextField;
13 import javafx.scene.control.Tooltip;
14 import javafx.scene.layout.BorderPane;
15 import javafx.scene.layout.GridPane;
16 import javafx.util.converter.NumberStringConverter;
17
18 public class UserInterfaceInputView extends BorderPane {
19
20  @FXML
21  private GridPane inputPane;
22
23  @FXML
24  private TextField projectSpanTF, numLeadersTF, numEngineersTF,
      agentTimeStepTF;
25
26  @FXML
27  private RadioButton EqualRadioBtn, DiffRadioBtn, tflOnBtn, tflOffBtn,
      sbaOnBtn, sbaOffBtn;
28
29  private DiversityApproach divApproach;
30
31  @FXML
32  private Button startBtn, continueBtn, resetBtn;
33
34  @FXML
35  private Label divApproachLabel;
36
37  @FXML
38  private ProgressBar progressBar;
39
40 // public ObjectProperty<DiversityModelSession> session;
41  public ObjectProperty<DiversityModelRunner> session;
42
43  public UserInterfaceInputView() {
```

```
44  FXMLLoader loader = new FXMLLoader(getClass().getResource("inputView.
     fxml"));
45  loader.setRoot(this);
46  loader.setController(this);
47  try {
48   loader.load();
49  } catch (IOException e1) {
50   e1.printStackTrace();
51  }
52
53  session = new SimpleObjectProperty<DiversityModelRunner>();
54
55  session.addListener((obs, oldV, newV) -> {
56   if (oldV != null) {
57    projectSpanTF.textProperty().unbindBidirectional(session.get().
     projectSpan);
58    numLeadersTF.textProperty().unbindBidirectional(session.get().
     numLeaders);
59    numEngineersTF.textProperty().unbindBidirectional(session.get().
     numEngineers);
60
61    tflOnBtn.selectedProperty().unbindBidirectional(session.get().
     transformationalLeader);
62    tflOffBtn.selectedProperty().unbindBidirectional(session.get().
     transformationalLeader);
63
64    sbaOnBtn.selectedProperty().unbindBidirectional(session.get().
     sociometricBadgeAwareness);
65    sbaOffBtn.selectedProperty().unbindBidirectional(session.get().
     sociometricBadgeAwareness);
66   }
67   if (newV != null) {
68    NumberStringConverter cv = new NumberStringConverter();
69    projectSpanTF.textProperty().bindBidirectional(session.get().
     projectSpan, cv);
70    numLeadersTF.textProperty().bindBidirectional(session.get().
     numLeaders, cv);
71    numEngineersTF.textProperty().bindBidirectional(session.get().
     numEngineers, cv);
72
73    if (session.get().transformationalLeader.get()) {
74     tflOnBtn.selectedProperty().bindBidirectional(session.get().
     transformationalLeader);
75    } else {
76     tflOffBtn.selectedProperty().bindBidirectional(session.get().
     transformationalLeader);
77    }
78
79    if (session.get().sociometricBadgeAwareness.get()) {
80     sbaOnBtn.selectedProperty().bindBidirectional(session.get().
     sociometricBadgeAwareness);
81    } else {
82     sbaOffBtn.selectedProperty().bindBidirectional(session.get().
     sociometricBadgeAwareness);
```

```
83      }
84
85     }
86    });
87
88    EqualRadioBtn.setOnAction(e -> {
89
90     session.get().setDiversityApproach(DiversityApproach.EQUALITY);
91  //   session.get().diversityApproach.set(DiversityApproach.EQUALITY);
92     DiffRadioBtn.setSelected(false);
93    });
94
95    DiffRadioBtn.setOnAction(e -> {
96     session.get().setDiversityApproach(DiversityApproach.DIFFERENCE);
97
98     // session.get().diversityApproach.set(DiversityApproach.DIFFERENCE);
99     EqualRadioBtn.setSelected(false);
100   });
101
102   tflOnBtn.setOnAction(e -> {
103    if (tflOnBtn.isSelected()) {
104     session.get().setTransformationalLeader(true);
105     tflOffBtn.setSelected(false);
106     sbaOnBtn.setDisable(false);
107    } else {
108     session.get().setTransformationalLeader(false);
109     tflOnBtn.setSelected(true);
110    }
111   });
112
113   tflOffBtn.setOnAction(e -> {
114    if (tflOffBtn.isSelected()) {
115     session.get().setTransformationalLeader(false);
116     session.get().setSociometricBadgeAwareness(false);
117     sbaOnBtn.setDisable(true);
118     sbaOffBtn.setSelected(true);
119     tflOnBtn.setSelected(false);
120    } else {
121     session.get().setTransformationalLeader(true);
122     tflOffBtn.setSelected(true);
123    }
124   });
125
126   sbaOnBtn.setOnAction(e -> {
127    if (sbaOnBtn.isSelected()) {
128     session.get().setSociometricBadgeAwareness(true);
129     sbaOffBtn.setSelected(false);
130    } else {
131     session.get().setSociometricBadgeAwareness(false);
132     sbaOnBtn.setSelected(true);
133    }
134   });
135
136   sbaOffBtn.setOnAction(e -> {
```

```
137     if (sbaOffBtn.isSelected()) {
138      session.get().setSociometricBadgeAwareness(false);
139      sbaOnBtn.setSelected(false);
140     } else {
141      session.get().setSociometricBadgeAwareness(true);
142      sbaOffBtn.setSelected(true);
143     }
144    });
145
146    startBtn.setOnAction(e -> {
147     session.get().start();
148     startBtn.setDisable(true);
149     continueBtn.setDisable(false);
150     continueBtn.setDefaultButton(true);
151    });
152
153    continueBtn.setOnAction(e -> {
154     session.get().agentEnvironment.continueSim();
155
156     // brute force method to stop sim when demands are complete
157 //    if (!session.get().agentEnvironment.checkAnyDemandList(session.get
        ().agentEnvironment.globalAssignedDemands)) {
158 //     continueBtn.setDisable(true);
159 //    };
160    });
161
162    resetBtn.setOnAction(e -> {
163     session.get().resetSim();
164     // this.session = new SimpleObjectProperty<DiversityModelRunnerFX>();
165     session.get().agentEnvironment.setGlobalTime(0);
166     startBtn.setDisable(false);
167     startBtn.setDefaultButton(true);
168     continueBtn.setDisable(true);
169    });
170
171    // Tooltips
172    Tooltip dvTooltip = new Tooltip();
173    dvTooltip.setText("Value in Equality vs. Value in Difference.  \n"
174      + "Value in Equality tend to give everyone equal opportunity, \n"
175      + "whereas Value in Difference approaches give someone a \n"
176      + "unique opportunity because of the skill sets he or she brings to
        the table.\n");
177    dvTooltip.install(divApproachLabel, dvTooltip);
178
179   }
180
181   // Getters and setters
182   public final ObjectProperty<DiversityModelRunner> sessionProperty() {
183    return this.session;
184   }
185
186   public final DiversityModelRunner getSession() {
187    return this.sessionProperty().get();
188   }
```

```
189
190  public final void setSession(final DiversityModelRunner session) {
191   this.sessionProperty().set(session);
192  }
193
194  public DiversityApproach getDivApproach() {
195   return divApproach;
196  }
197
198  public void setDivApproach(DiversityApproach divApproach) {
199   this.divApproach = divApproach;
200  }
201
202 }
```

A.1.3 UserInterfaceResultsView.java

```
1
2  import java.io.File;
3  import java.io.IOException;
4  import java.time.LocalDateTime;
5  import java.time.format.DateTimeFormatter;
6  import java.util.List;
7
8  import javafx.beans.property.ListProperty;
9  import javafx.beans.property.ObjectProperty;
10 import javafx.beans.property.SimpleListProperty;
11 import javafx.beans.property.SimpleObjectProperty;
12 import javafx.collections.FXCollections;
13 import javafx.collections.ListChangeListener;
14 import javafx.fxml.FXML;
15 import javafx.fxml.FXMLLoader;
16 import javafx.scene.chart.LineChart;
17 import javafx.scene.chart.NumberAxis;
18 import javafx.scene.chart.XYChart;
19 import javafx.scene.chart.XYChart.Data;
20 import javafx.scene.chart.XYChart.Series;
21 import javafx.scene.control.Button;
22 import javafx.scene.control.TextField;
23 import javafx.scene.control.Tooltip;
24 import javafx.scene.layout.BorderPane;
25 import javafx.stage.FileChooser;
26 import javafx.util.converter.NumberStringConverter;
27
28 public class UserInterfaceResultsView extends BorderPane {
29
30  // UI Properties
31  @FXML
32  private LineChart<Number, Number> retentionChart, demandsChart,
       diversityChart, internalDensityChart;
33  @FXML
34  private NumberAxis xAxisRetention, yAxisRetention;
35
36  @FXML
```

```
37  private NumberAxis xAxisDemands, yAxisDemands;
38
39  @FXML
40  private NumberAxis xAxisCommitment, yAxisEngagement;
41  @FXML
42  private TextField globalTimeTF;
43  @FXML
44  private TextField globalDemandsMootTF, globalDiversityDemandsMootTF;
45  @FXML
46  private TextField agentsLostLowTF, agentsLostModTF, agentsLostHighTF;
47  @FXML
48  private TextField indStartTF, indFinishTF;
49  @FXML
50  private Button saveBtn;
51
52  // Collections of results
53  private ListProperty<Data<Number, Number>> lowRepAgentsRetained;
54  private ListChangeListener<Data<Number, Number>>
      lowRepAgentsRetainedListener;
55  private XYChart.Series<Number, Number> lowRepAgentsRetainedSeries;
56
57  private ListProperty<Data<Number, Number>> modRepAgentsRetained;
58  private ListChangeListener<Data<Number, Number>>
      modRepAgentsRetainedListener;
59  private XYChart.Series<Number, Number> modRepAgentsRetainedSeries;
60
61  private ListProperty<Data<Number, Number>> highRepAgentsRetained;
62  private ListChangeListener<Data<Number, Number>>
      highRepAgentsRetainedListener;
63  private XYChart.Series<Number, Number> highRepAgentsRetainedSeries;
64
65  private ListProperty<Data<Number, Number>> demandsRemaining;
66  private ListChangeListener<Data<Number, Number>>
      demandsRemainingListener;
67  private XYChart.Series<Number, Number> demandsRemainingSeries;
68
69  private ListProperty<Data<Number, Number>> coordRequestsComplete;
70  private ListChangeListener<Data<Number, Number>> coordRequestsListener;
71  private XYChart.Series<Number, Number> coordRequestsSeries;
72
73  private ListProperty<Data<Number, Number>> leadershipTasks;
74  private ListChangeListener<Data<Number, Number>>
      leadershipTasksListener;
75  private XYChart.Series<Number, Number> leadershipTasksSeries;
76
77  private ListProperty<Data<Number, Number>>
      diversityConsequencesGenerated;
78  private ListChangeListener<Data<Number, Number>>
      diversityConsequencesListener;
79  private XYChart.Series<Number, Number> diversityConcernsSeries;
80
81  private ListProperty<Data<Number, Number>> agentsAbsent;
82  private ListChangeListener<Data<Number, Number>> agentsAbsentListener;
83  private XYChart.Series<Number, Number> agentsAbsentSeries;
```

```
84
85  private ListProperty<Data<Number, Number>> networkDensities;
86  private ListChangeListener<Data<Number, Number>>
      teamInternalNetworkDensityListener;
87  private XYChart.Series<Number, Number> networkDensitiesSeries;
88
89  private ListProperty<Data<Number, Number>> networkDensitiesLow;
90  private ListChangeListener<Data<Number, Number>>
      teamInternalNetworkDensityLowListener;
91  private XYChart.Series<Number, Number> networkDensitiesSeriesLow;
92
93  private ListProperty<Data<Number, Number>> networkDensitiesMod;
94  private ListChangeListener<Data<Number, Number>>
      teamInternalNetworkDensityModListener;
95  private XYChart.Series<Number, Number> networkDensitiesSeriesMod;
96
97  private ListProperty<Data<Number, Number>> networkDensitiesHigh;
98  private ListChangeListener<Data<Number, Number>>
      teamInternalNetworkDensityHighListener;
99  private XYChart.Series<Number, Number> networkDensitiesSeriesHigh;
100
101 // simulator
102 public SimpleObjectProperty<DiversityModelRunner> session;
103
104 // Constructor
105 public UserInterfaceResultsView() {
106  FXMLLoader loader = new FXMLLoader(getClass().getResource("ResultsView
       .fxml"));
107  loader.setRoot(this);
108  loader.setController(this);
109  try {
110   loader.load();
111  } catch (IOException e1) {
112   e1.printStackTrace();
113  }
114
115  session = new SimpleObjectProperty<DiversityModelRunner>();
116
117  // Establish listeners for sim environment results
118  // Listen for when an agent goes dormant
119  lowRepAgentsRetainedListener = new ListChangeListener<Data<Number,
       Number>>() {
120   @Override
121   public void onChanged(javafx.collections.ListChangeListener.Change<?
      extends Data<Number, Number>> c) {
122
123    while (c.next()) {
124     if (c.wasAdded()) {
125      List<? extends Data<Number, Number>> addedSubList = c.
     getAddedSubList();
126       for (Data<Number, Number> x : addedSubList) {
127        addResultToChart(lowRepAgentsRetainedSeries, x);
128       }
129     } else if (c.wasRemoved()) {
```

```
130         lowRepAgentsRetainedSeries.getData().clear();
131         break;
132        }
133       }
134      }
135     };
136
137     modRepAgentsRetainedListener = new ListChangeListener<Data<Number,
        Number>>() {
138      @Override
139      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
140
141       while (c.next()) {
142        if (c.wasAdded()) {
143         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
144         for (Data<Number, Number> x : addedSubList) {
145          addResultToChart(modRepAgentsRetainedSeries, x);
146         }
147        } else if (c.wasRemoved()) {
148         modRepAgentsRetainedSeries.getData().clear();
149         break;
150        }
151       }
152      }
153     };
154
155     highRepAgentsRetainedListener = new ListChangeListener<Data<Number,
        Number>>() {
156      @Override
157      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
158
159       while (c.next()) {
160        if (c.wasAdded()) {
161         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
162         for (Data<Number, Number> x : addedSubList) {
163          addResultToChart(highRepAgentsRetainedSeries, x);
164         }
165        } else if (c.wasRemoved()) {
166         highRepAgentsRetainedSeries.getData().clear();
167         break;
168        }
169       }
170      }
171     };
172
173     // Listen for when demands are completed
174     demandsRemainingListener = new ListChangeListener<Data<Number, Number
        >>() {
175      @Override
176      public void onChanged(javafx.collections.ListChangeListener.Change<?
```

```
 extends Data<Number, Number>> c) {
177
178     while (c.next()) {
179       if (c.wasAdded()) {
180         List<? extends Data<Number, Number>> addedSubList = c.
      getAddedSubList();
181         for (Data<Number, Number> x : addedSubList) {
182           addResultToChart(demandsRemainingSeries, x);
183         }
184       } else if (c.wasRemoved()) {
185         demandsRemainingSeries.getData().clear();
186         break;
187       }
188     }
189   }
190 };
191
192 // Listen for when coordinated demands are completed
193 coordRequestsListener = new ListChangeListener<Data<Number, Number>>()
      {
194   @Override
195   public void onChanged(javafx.collections.ListChangeListener.Change<?
      extends Data<Number, Number>> c) {
196
197     while (c.next()) {
198       if (c.wasAdded()) {
199         List<? extends Data<Number, Number>> addedSubList = c.
      getAddedSubList();
200         for (Data<Number, Number> x : addedSubList) {
201           addResultToChart(coordRequestsSeries, x);
202         }
203       } else if (c.wasRemoved()) {
204         coordRequestsSeries.getData().clear();
205         break;
206       }
207     }
208   }
209 };
210
211 // Listen for when leader addresses diversity demands
212 leadershipTasksListener = new ListChangeListener<Data<Number, Number
      >>() {
213   @Override
214   public void onChanged(javafx.collections.ListChangeListener.Change<?
      extends Data<Number, Number>> c) {
215
216     while (c.next()) {
217       if (c.wasAdded()) {
218         List<? extends Data<Number, Number>> addedSubList = c.
      getAddedSubList();
219         for (Data<Number, Number> x : addedSubList) {
220           addResultToChart(leadershipTasksSeries, x);
221         }
222       } else if (c.wasRemoved()) {
```

```
223         leadershipTasksSeries.getData().clear();
224         break;
225        }
226       }
227      }
228     };
229
230     // Listen for when diversity-related consequences are generated
231     diversityConsequencesListener = new ListChangeListener<Data<Number,
        Number>>() {
232      @Override
233      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
234
235       while (c.next()) {
236        if (c.wasAdded()) {
237         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
238         for (Data<Number, Number> x : addedSubList) {
239          addResultToChart(diversityConcernsSeries, x);
240         }
241        } else if (c.wasRemoved()) {
242         diversityConcernsSeries.getData().clear();
243         break;
244        }
245       }
246      }
247     };
248
249     // Listen for when diversity-related consequences are generated
250     agentsAbsentListener = new ListChangeListener<Data<Number, Number>>()
        {
251      @Override
252      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
253
254       while (c.next()) {
255        if (c.wasAdded()) {
256         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
257         for (Data<Number, Number> x : addedSubList) {
258          addResultToChart(agentsAbsentSeries, x);
259         }
260        } else if (c.wasRemoved()) {
261         agentsAbsentSeries.getData().clear();
262         break;
263        }
264       }
265      }
266     };
267
268     // Listen for when team's internal network density changes
269     teamInternalNetworkDensityListener = new ListChangeListener<Data<
        Number, Number>>() {
```

```
270     @Override
271     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
272
273      while (c.next()) {
274       if (c.wasAdded()) {
275        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
276         for (Data<Number, Number> x : addedSubList) {
277          addResultToChart(networkDensitiesSeries, x);
278         }
279       } else if (c.wasRemoved()) {
280        networkDensitiesSeries.getData().clear();
281        break;
282       }
283      }
284     }
285    };
286
287    teamInternalNetworkDensityLowListener = new ListChangeListener<Data<
        Number, Number>>() {
288     @Override
289     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
290
291      while (c.next()) {
292       if (c.wasAdded()) {
293        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
294         for (Data<Number, Number> x : addedSubList) {
295          addResultToChart(networkDensitiesSeriesLow, x);
296         }
297       } else if (c.wasRemoved()) {
298        networkDensitiesSeriesLow.getData().clear();
299        break;
300       }
301      }
302     }
303    };
304
305    teamInternalNetworkDensityModListener = new ListChangeListener<Data<
        Number, Number>>() {
306     @Override
307     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
308
309      while (c.next()) {
310       if (c.wasAdded()) {
311        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
312         for (Data<Number, Number> x : addedSubList) {
313          addResultToChart(networkDensitiesSeriesMod, x);
314         }
315       } else if (c.wasRemoved()) {
```

```
316         networkDensitiesSeriesMod.getData().clear();
317         break;
318        }
319       }
320      }
321     };
322
323     teamInternalNetworkDensityHighListener = new ListChangeListener<Data<
        Number, Number>>() {
324      @Override
325      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
326
327       while (c.next()) {
328        if (c.wasAdded()) {
329         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
330         for (Data<Number, Number> x : addedSubList) {
331          addResultToChart(networkDensitiesSeriesHigh, x);
332         }
333        } else if (c.wasRemoved()) {
334         networkDensitiesSeriesHigh.getData().clear();
335         break;
336        }
337       }
338      }
339     };
340
341     // Initialize lists for storing results and add binding listener to
        them
342     lowRepAgentsRetained = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
343     lowRepAgentsRetained.addListener(lowRepAgentsRetainedListener);
344
345     modRepAgentsRetained = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
346     modRepAgentsRetained.addListener(modRepAgentsRetainedListener);
347
348     highRepAgentsRetained = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
349     highRepAgentsRetained.addListener(highRepAgentsRetainedListener);
350
351     demandsRemaining = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
352     demandsRemaining.addListener(demandsRemainingListener);
353
354     coordRequestsComplete = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
355     coordRequestsComplete.addListener(coordRequestsListener);
356
357     leadershipTasks = new SimpleListProperty<Data<Number, Number>>(
        FXCollections.observableArrayList());
358     leadershipTasks.addListener(leadershipTasksListener);
359
```

```
360   diversityConsequencesGenerated = new SimpleListProperty<Data<Number,
       Number>>(
361     FXCollections.observableArrayList());
362   diversityConsequencesGenerated.addListener(
       diversityConsequencesListener);
363
364   agentsAbsent = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
365   agentsAbsent.addListener(agentsAbsentListener);
366
367   networkDensities = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
368   networkDensities.addListener(teamInternalNetworkDensityListener);
369
370   networkDensitiesLow = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
371   networkDensitiesLow.addListener(teamInternalNetworkDensityLowListener)
       ;
372
373   networkDensitiesMod = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
374   networkDensitiesMod.addListener(teamInternalNetworkDensityModListener)
       ;
375
376   networkDensitiesHigh = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
377   networkDensitiesHigh.addListener(
       teamInternalNetworkDensityHighListener);
378
379   // Establish a listener for the simulation to update User Interface
380   session.addListener((obs, oldV, newV) -> {
381
382    if (oldV != null) {
383     lowRepAgentsRetained.unbind();
384     modRepAgentsRetained.unbind();
385     highRepAgentsRetained.unbind();
386     demandsRemaining.unbind();
387     coordRequestsComplete.unbind();
388     leadershipTasks.unbind();
389
390     diversityConsequencesGenerated.unbind();
391     networkDensities.unbind();
392     networkDensitiesLow.unbind();
393     networkDensitiesMod.unbind();
394     networkDensitiesHigh.unbind();
395     globalTimeTF.textProperty().unbind();
396     globalTimeTF.textProperty().unbindBidirectional(session.get().
       agentEnvironment.globalTime);
397
398     globalDemandsMootTF.textProperty()
399       .unbindBidirectional(session.get().agentEnvironment.
       n_completeDemands);
400
401    }
```

```
402
403     if (newV != null) {
404      NumberStringConverter cv = new NumberStringConverter();
405
406      // Update data for charts
407      lowRepAgentsRetained.bind(session.get().agentEnvironment.
        lowRepAgentsRetained_timeSeriesProperty());
408      modRepAgentsRetained.bind(session.get().agentEnvironment.
        modRepAgentsRetained_timeSeriesProperty());
409      highRepAgentsRetained.bind(session.get().agentEnvironment.
        highRepAgentsRetained_timeSeriesProperty());
410      demandsRemaining.bind(session.get().agentEnvironment.
        completedDemands_timeSeriesProperty());
411      coordRequestsComplete.bind(session.get().agentEnvironment.
        coordRequests_timeSeriesProperty());
412      leadershipTasks.bind(session.get().agentEnvironment.
        leadershipTasksComplete_timeSeriesProperty());
413      diversityConsequencesGenerated
414        .bind(session.get().agentEnvironment.
        diversityConcerns_timeSeriesProperty());
415      agentsAbsent.bind(session.get().agentEnvironment.
        agentsAbsent_timeSeriesProperty());
416      networkDensities.bind(session.get().agentEnvironment.
        intNetDensity_timeSeriesProperty());
417      networkDensitiesLow.bind(session.get().agentEnvironment.
        intNetDensityLow_timeSeriesProperty());
418      networkDensitiesMod.bind(session.get().agentEnvironment.
        intNetDensityMod_timeSeriesProperty());
419      networkDensitiesHigh.bind(session.get().agentEnvironment.
        intNetDensityHigh_timeSeriesProperty());
420
421      // Update text fields
422      globalTimeTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.globalTimeDays, cv);
423      globalDemandsMootTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_completeDemands,
424        cv);
425      globalDiversityDemandsMootTF.textProperty()
426        .bindBidirectional(session.get().agentEnvironment.
        n_completeDiversityDemands, cv);
427      agentsLostLowTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_low, cv);
428      agentsLostModTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_moderate,
429        cv);
430      agentsLostHighTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_high, cv);
431      indStartTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.indStart, cv);
432      indFinishTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.indFinish, cv);
433     }
434    });
435
```

```
436   // Initialize charts & add data to chart
437
438   // Task Progress Chart
439   demandsRemainingSeries = new XYChart.Series<Number, Number>();
440   demandsRemainingSeries.setName("Demands Remaining");
441   demandsChart.getData().add(demandsRemainingSeries);
442
443   coordRequestsSeries = new XYChart.Series<Number, Number>();
444   coordRequestsSeries.setName("Interactions");
445   demandsChart.getData().add(coordRequestsSeries);
446
447   // Diversity Consequences Chart
448   diversityConcernsSeries = new XYChart.Series<Number, Number>();
449   diversityConcernsSeries.setName("Lingering Diversity Concerns");
450   diversityConcernsSeries.getData().add(new Data<Number, Number>(0, 0));
451   diversityChart.getData().add(diversityConcernsSeries);
452
453   agentsAbsentSeries = new XYChart.Series<Number, Number>();
454   agentsAbsentSeries.setName("Agent Absences");
455   agentsAbsentSeries.getData().add(new Data<Number, Number>(0, 0));
456   diversityChart.getData().add(agentsAbsentSeries);
457
458   leadershipTasksSeries = new XYChart.Series<Number, Number>();
459   leadershipTasksSeries.setName("Leader-mitigated Diversity Concerns");
460   diversityChart.getData().add(leadershipTasksSeries);
461
462   // Retention Chart
463   highRepAgentsRetainedSeries = new XYChart.Series<Number, Number>();
464   highRepAgentsRetainedSeries.setName("High Representation");
465   retentionChart.getData().add(highRepAgentsRetainedSeries);
466
467   modRepAgentsRetainedSeries = new XYChart.Series<Number, Number>();
468   modRepAgentsRetainedSeries.setName("Moderate Representation");
469   retentionChart.getData().add(modRepAgentsRetainedSeries);
470
471   lowRepAgentsRetainedSeries = new XYChart.Series<Number, Number>();
472   lowRepAgentsRetainedSeries.setName("Low Representation");
473   retentionChart.getData().add(lowRepAgentsRetainedSeries);
474
475   // Internal Density Chart
476   networkDensitiesSeriesHigh = new XYChart.Series<Number, Number>();
477   networkDensitiesSeriesHigh.setName("High Representation");
478   internalDensityChart.getData().add(networkDensitiesSeriesHigh);
479
480   networkDensitiesSeriesMod = new XYChart.Series<Number, Number>();
481   networkDensitiesSeriesMod.setName("Moderate Representation");
482   internalDensityChart.getData().add(networkDensitiesSeriesMod);
483
484   networkDensitiesSeriesLow = new XYChart.Series<Number, Number>();
485   networkDensitiesSeriesLow.setName("Low Representation");
486   internalDensityChart.getData().add(networkDensitiesSeriesLow);
487
488   networkDensitiesSeries = new XYChart.Series<Number, Number>();
489   networkDensitiesSeries.setName("Team");
```

```
490   internalDensityChart.getData().add(networkDensitiesSeries);
491
492   this.saveBtn.setOnAction(e -> {
493    saveResults();
494   });
495  }
496
497  public final ObjectProperty<DiversityModelRunner> sessionProperty() {
498   return this.session;
499  }
500
501  public final DiversityModelRunner getSession() {
502   return this.sessionProperty().get();
503  }
504
505  public final void setSession(final DiversityModelRunner session) {
506   this.sessionProperty().set(session);
507  }
508
509  private void addResultToChart(Series<Number, Number> dataSeries, Data<
       Number, Number> x) {
510   var d = new XYChart.Data<Number, Number>(x.getXValue(), x.getYValue())
       ;
511   d.setExtraValue(x);
512   dataSeries.getData().add(d);
513   Tooltip tooltip = new Tooltip();
514   tooltip.setText("" + x.getXValue() + ", " + x.getYValue());
515   Tooltip.install(d.getNode(), tooltip);
516  }
517
518  public void saveResults() {
519   ListProperty<SimResult> dummyList = new SimpleListProperty<>(
       FXCollections.observableArrayList());
520   dummyList.add(session.get().simResults);
521
522   DateTimeFormatter dtf = DateTimeFormatter.ofPattern(" yyyy-MM-
       dd_HHmmss");
523   LocalDateTime now = LocalDateTime.now();
524   var chooserOut = new FileChooser();
525   chooserOut.setTitle("Please save the results to a file");
526   chooserOut.setInitialFileName("DiversityABM_ResultData_" + dtf.format(
       now) + ".xlsx");
527   chooserOut.getExtensionFilters().addAll(new FileChooser.
       ExtensionFilter("Excel Workbook Files", "*.xlsx"));
528
529   File fileToWrite = chooserOut.showSaveDialog(null);
530
531   var re = new ResultsExporter(dummyList);
532   re.export(fileToWrite);
533  }
534 }
```

A.1.4 inputView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.RadioButton?>
<?import javafx.scene.control.Separator?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>

<fx:root maxHeight="500.0" maxWidth="300.0" minHeight="500.0" minWidth="
    300.0" prefHeight="500.0" prefWidth="300.0" type="BorderPane" xmlns=
    "http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1">
   <top>
      <Label alignment="CENTER" minWidth="400.0" prefHeight="39.0"
    prefWidth="540.0" text="Simulation Inputs" BorderPane.alignment="
    CENTER">
         <font>
            <Font size="32.0" />
         </font>
      </Label>
   </top>
   <center>
      <GridPane fx:id="inputPane" BorderPane.alignment="CENTER">
        <columnConstraints>
          <ColumnConstraints halignment="RIGHT" hgrow="ALWAYS" />
          <ColumnConstraints halignment="CENTER" hgrow="ALWAYS" minWidth
    ="0.0" />
            <ColumnConstraints />
        </columnConstraints>
        <rowConstraints>
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints percentHeight="-2.0" vgrow="SOMETIMES" />
            <RowConstraints maxHeight="30.5" minHeight="10.0"
    percentHeight="0.0" prefHeight="11.0" vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints percentHeight="0.0" vgrow="SOMETIMES" />
            <RowConstraints percentHeight="0.0" vgrow="SOMETIMES" />
            <RowConstraints percentHeight="0.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" percentHeight="0.0"
    prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
            <Label alignment="CENTER" contentDisplay="CENTER" text="
```

```
Project Duration (days)" wrapText="true" GridPane.rowIndex="7" />
48          <TextField fx:id="projectSpanTF" GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="7">
49              <GridPane.margin>
50                  <Insets bottom="10.0" left="10.0" right="10.0" top="
10.0" />
51              </GridPane.margin>
52          </TextField>
53          <TextField fx:id="numLeadersTF" GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="8">
54              <GridPane.margin>
55                  <Insets bottom="10.0" left="10.0" right="10.0" top="
10.0" />
56              </GridPane.margin>
57          </TextField>
58          <TextField fx:id="numEngineersTF" GridPane.columnIndex="1"
GridPane.columnSpan="2" GridPane.rowIndex="9">
59              <GridPane.margin>
60                  <Insets bottom="10.0" left="10.0" right="10.0" top="
10.0" />
61              </GridPane.margin>
62          </TextField>
63          <Label text="# of Formal Leaders" wrapText="true" GridPane.
rowIndex="8" />
64          <Label text="# of Project Engineers" wrapText="true"
GridPane.rowIndex="9" />
65          <Label fx:id="divApproachLabel" text="Diversity Value
Approach" textAlignment="CENTER" wrapText="true" GridPane.columnSpan
="2" GridPane.halignment="CENTER" />
66          <Button fx:id="startBtn" defaultButton="true" minWidth="-
Infinity" mnemonicParsing="false" prefWidth="75.0" text="Start"
GridPane.halignment="CENTER" GridPane.rowIndex="10">
67              <GridPane.margin>
68                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
69              </GridPane.margin></Button>
70          <Button fx:id="continueBtn" disable="true" minWidth="-
Infinity" mnemonicParsing="false" prefWidth="75.0" text="Continue"
GridPane.columnIndex="1" GridPane.rowIndex="10">
71              <GridPane.margin>
72                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
73              </GridPane.margin></Button>
74          <Button fx:id="resetBtn" minWidth="-Infinity"
mnemonicParsing="false" prefWidth="75.0" text="Reset" GridPane.
columnSpan="2147483647" GridPane.halignment="CENTER" GridPane.
rowIndex="12">
75              <GridPane.margin>
76                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
77              </GridPane.margin></Button>
78          <RadioButton fx:id="EqualRadioBtn" minWidth="-Infinity"
mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
90.0" selected="true" text="Equality" GridPane.halignment="CENTER"
```

```
    GridPane.rowIndex="1">
79                <GridPane.margin>
80                   <Insets bottom="10.0" left="10.0" right="10.0" />
81                </GridPane.margin></RadioButton>
82             <RadioButton fx:id="DiffRadioBtn" minWidth="-Infinity"
    mnemonicParsing="false" prefWidth="90.0" text="Difference" GridPane.
    columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="1">
83                <GridPane.margin>
84                   <Insets bottom="10.0" left="10.0" right="10.0" />
85                </GridPane.margin></RadioButton>
86             <RadioButton fx:id="tflOnBtn" minWidth="-Infinity"
    mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
    90.0" selected="true" text="On" GridPane.halignment="CENTER"
    GridPane.rowIndex="3">
87                <GridPane.margin>
88                   <Insets bottom="10.0" left="10.0" right="10.0" />
89                </GridPane.margin>
90             </RadioButton>
91             <RadioButton fx:id="tflOffBtn" minWidth="-Infinity"
    mnemonicParsing="false" prefWidth="90.0" text="Off" GridPane.
    columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="3">
92                <GridPane.margin>
93                   <Insets bottom="10.0" left="10.0" right="10.0" />
94                </GridPane.margin>
95             </RadioButton>
96             <Label text="Transformational Leadership" textAlignment="
    CENTER" wrapText="true" GridPane.columnSpan="2" GridPane.halignment=
    "CENTER" GridPane.rowIndex="2" />
97             <Label text="Sociometric Badge Awareness" textAlignment="
    CENTER" wrapText="true" GridPane.columnSpan="2" GridPane.halignment=
    "CENTER" GridPane.rowIndex="4" />
98             <RadioButton fx:id="sbaOnBtn" minWidth="-Infinity"
    mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
    90.0" selected="true" text="On" GridPane.halignment="CENTER"
    GridPane.rowIndex="5">
99                <GridPane.margin>
100                  <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
101               </GridPane.margin>
102            </RadioButton>
103            <RadioButton fx:id="sbaOffBtn" minWidth="-Infinity"
    mnemonicParsing="false" prefWidth="90.0" text="Off" GridPane.
    columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="5">
104               <GridPane.margin>
105                  <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
106               </GridPane.margin>
107            </RadioButton>
108            <Separator prefWidth="200.0" GridPane.columnSpan="2147483647
    " GridPane.rowIndex="6" />
109         </children>
110      </GridPane>
111   </center>
112 </fx:root>
```

A.1.5 ResultsView.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.chart.LineChart?>
<?import javafx.scene.chart.NumberAxis?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Separator?>
<?import javafx.scene.control.SplitPane?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>

<fx:root type="BorderPane" xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1">
   <top>
      <Label text="Single Simulation Results" BorderPane.alignment="CENTER">
         <font>
            <Font size="26.0" />
         </font>
      </Label>
   </top>
   <center>
      <GridPane BorderPane.alignment="CENTER">
         <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" />
            <ColumnConstraints hgrow="SOMETIMES" />
         </columnConstraints>
         <rowConstraints>
            <RowConstraints vgrow="SOMETIMES" />
            <RowConstraints maxHeight="415.0" minHeight="10.0" prefHeight="376.0" vgrow="SOMETIMES" />
         </rowConstraints>
         <children>
            <SplitPane dividerPositions="0.5229276895943563" prefHeight="160.0" prefWidth="200.0" GridPane.columnSpan="2147483647" GridPane.rowSpan="2147483647">
              <items>
                  <SplitPane dividerPositions="0.46234309623430964" orientation="VERTICAL">
                    <items>
                       <LineChart fx:id="demandsChart" animated="false" style="-fx-border-color: a31f34;" stylesheets="@application/application2.css" title="Task Progress">
                         <xAxis>
                           <NumberAxis label="Time (Days)" side="BOTTOM" fx:id="xAxisDemands" />
                         </xAxis>
```

```
                        <yAxis>
                          <NumberAxis fx:id="yAxisDemands" label="
Tasks" side="LEFT" />
                        </yAxis>
                      </LineChart>
                      <LineChart fx:id="diversityChart" animated="
false" style="-fx-border-color: a31f34;" stylesheets="@application/
application2.css" title="Diversity Consequences">
                        <xAxis>
                          <NumberAxis label="Time (Days)" side="BOTTOM
" fx:id="xAxisCommitment" />
                        </xAxis>
                        <yAxis>
                          <NumberAxis fx:id="yAxisEngagement" label="
Number" side="LEFT" />
                        </yAxis>
                      </LineChart>
                    </items>
                  </SplitPane>
                  <SplitPane dividerPositions="0.46373779637377965"
orientation="VERTICAL" prefHeight="200.0" prefWidth="160.0">
                    <items>
                      <LineChart fx:id="internalDensityChart" animated
="false" style="-fx-border-color: a31f34;" stylesheets="@application
/application2.css" title="Average Team Int Net Density">
                        <xAxis>
                          <NumberAxis label="Time (Days)" side="BOTTOM
" />
                        </xAxis>
                        <yAxis>
                          <NumberAxis lowerBound="0.25" side="LEFT"
upperBound="1.0" visible="false" />
                        </yAxis>
                      </LineChart>
                      <LineChart fx:id="retentionChart" animated="
false" prefHeight="295.0" prefWidth="428.0" style="-fx-border-color:
 a31f34;" stylesheets="@application/application2.css" title="
Retention">
                        <xAxis>
                          <NumberAxis label="Time (days)" side="BOTTOM
" fx:id="xAxisRetention" />
                        </xAxis>
                        <yAxis>
                          <NumberAxis fx:id="yAxisRetention" label="
Agents Retained" side="LEFT" />
                        </yAxis>
                      </LineChart>
                    </items>
                  </SplitPane>
                </items>
              </SplitPane>
            </children>
          </GridPane>
        </center>
```

```
83    <right>
84       <GridPane BorderPane.alignment="TOP_CENTER">
85          <columnConstraints>
86             <ColumnConstraints hgrow="SOMETIMES" />
87             <ColumnConstraints hgrow="SOMETIMES" />
88          </columnConstraints>
89          <rowConstraints>
90             <RowConstraints vgrow="SOMETIMES" />
91             <RowConstraints vgrow="SOMETIMES" />
92             <RowConstraints vgrow="SOMETIMES" />
93             <RowConstraints vgrow="SOMETIMES" />
94             <RowConstraints vgrow="SOMETIMES" />
95             <RowConstraints vgrow="SOMETIMES" />
96             <RowConstraints vgrow="SOMETIMES" />
97             <RowConstraints vgrow="SOMETIMES" />
98             <RowConstraints vgrow="SOMETIMES" />
99             <RowConstraints valignment="BOTTOM" vgrow="SOMETIMES" />
100            <RowConstraints valignment="TOP" vgrow="SOMETIMES" />
101            <RowConstraints valignment="TOP" vgrow="SOMETIMES" />
102            <RowConstraints maxHeight="22.0" minHeight="8.0" prefHeight=
       "22.0" valignment="TOP" vgrow="SOMETIMES" />
103            <RowConstraints maxHeight="25.0" minHeight="10.0" prefHeight
       ="21.0" valignment="TOP" vgrow="SOMETIMES" />
104            <RowConstraints maxHeight="26.0" minHeight="10.0" prefHeight
       ="20.0" valignment="TOP" vgrow="SOMETIMES" />
105            <RowConstraints valignment="TOP" vgrow="SOMETIMES" />
106         </rowConstraints>
107         <children>
108            <Label text="Total Time (Days)" GridPane.halignment="RIGHT"
       GridPane.rowIndex="1">
109               <GridPane.margin>
110                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
       />
111               </GridPane.margin>
112            </Label>
113            <TextField fx:id="globalTimeTF" editable="false" prefWidth="
       80.0" GridPane.columnIndex="1" GridPane.rowIndex="1">
114               <GridPane.margin>
115                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
       />
116               </GridPane.margin>
117            </TextField>
118            <Label text="Demands Completed" underline="true" GridPane.
       columnSpan="2147483647" GridPane.halignment="CENTER" GridPane.
       rowIndex="2" GridPane.valignment="BOTTOM">
119               <GridPane.margin>
120                  <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
       />
121               </GridPane.margin>
122            </Label>
123            <Label alignment="CENTER" contentDisplay="CENTER" text="
       Outputs" textAlignment="CENTER" GridPane.columnSpan="2147483647"
       GridPane.halignment="CENTER">
124               <GridPane.margin>
```

```
125                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
126             </GridPane.margin>
127             <font>
128                <Font size="22.0" />
129             </font>
130          </Label>
131          <Label text="Project" GridPane.halignment="RIGHT" GridPane.
    rowIndex="3" GridPane.valignment="TOP">
132             <GridPane.margin>
133                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
134             </GridPane.margin>
135          </Label>
136          <Label text="Diversity" GridPane.halignment="RIGHT" GridPane
    .rowIndex="4" GridPane.valignment="TOP">
137             <GridPane.margin>
138                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
139             </GridPane.margin>
140          </Label>
141          <Label alignment="CENTER" contentDisplay="CENTER" text="
    Project Staff Lost" underline="true" GridPane.columnSpan="2"
    GridPane.halignment="CENTER" GridPane.rowIndex="8" GridPane.
    valignment="BOTTOM">
142             <GridPane.margin>
143                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
144             </GridPane.margin>
145          </Label>
146          <Label text="Low Representation" GridPane.halignment="RIGHT"
     GridPane.rowIndex="9" GridPane.valignment="TOP">
147             <GridPane.margin>
148                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
149             </GridPane.margin>
150          </Label>
151          <Label text="Moderate Representation" GridPane.halignment="
    RIGHT" GridPane.rowIndex="10" GridPane.valignment="TOP">
152             <GridPane.margin>
153                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
154             </GridPane.margin>
155          </Label>
156          <Label text="Average Internal Network Density" underline="
    true" GridPane.columnSpan="2" GridPane.halignment="CENTER" GridPane.
    rowIndex="5" GridPane.valignment="BOTTOM">
157             <GridPane.margin>
158                <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
159             </GridPane.margin>
160          </Label>
161          <Label text="Finishing Internal Density" GridPane.halignment
    ="RIGHT" GridPane.rowIndex="7">
```

```
162                 <GridPane.margin>
163                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
164                 </GridPane.margin>
165             </Label>
166             <Label text="High Representation" GridPane.halignment="RIGHT
    " GridPane.rowIndex="11" GridPane.valignment="TOP">
167                 <GridPane.margin>
168                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
169                 </GridPane.margin>
170             </Label>
171             <Label text="Starting Internal Density" GridPane.halignment=
    "RIGHT" GridPane.rowIndex="6">
172                 <GridPane.margin>
173                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
174                 </GridPane.margin>
175             </Label>
176             <TextField fx:id="globalDemandsMootTF" editable="false"
    prefWidth="80.0" GridPane.columnIndex="1" GridPane.rowIndex="3"
    GridPane.valignment="TOP">
177                 <GridPane.margin>
178                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
179                 </GridPane.margin>
180             </TextField>
181             <TextField fx:id="globalDiversityDemandsMootTF" editable="
    false" prefWidth="80.0" GridPane.columnIndex="1" GridPane.rowIndex="
    4" GridPane.valignment="TOP">
182                 <GridPane.margin>
183                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
184                 </GridPane.margin>
185             </TextField>
186             <TextField fx:id="agentsLostLowTF" editable="false"
    prefWidth="80.0" GridPane.columnIndex="1" GridPane.rowIndex="9"
    GridPane.valignment="TOP">
187                 <GridPane.margin>
188                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
189                 </GridPane.margin>
190             </TextField>
191             <TextField fx:id="agentsLostModTF" editable="false"
    prefWidth="80.0" GridPane.columnIndex="1" GridPane.rowIndex="10"
    GridPane.valignment="TOP">
192                 <GridPane.margin>
193                     <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
194                 </GridPane.margin>
195             </TextField>
196             <TextField fx:id="agentsLostHighTF" editable="false"
    prefWidth="80.0" GridPane.columnIndex="1" GridPane.rowIndex="11"
    GridPane.valignment="TOP">
```

```
197                <GridPane.margin>
198                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
199                </GridPane.margin>
200             </TextField>
201             <TextField fx:id="indStartTF" editable="false" prefWidth="
    80.0" GridPane.columnIndex="1" GridPane.rowIndex="6">
202                <GridPane.margin>
203                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
204                </GridPane.margin>
205             </TextField>
206             <TextField fx:id="indFinishTF" editable="false" prefWidth="
    80.0" GridPane.columnIndex="1" GridPane.rowIndex="7">
207                <GridPane.margin>
208                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
209                </GridPane.margin>
210             </TextField>
211             <Separator prefWidth="200.0" GridPane.columnSpan="2147483647
    " GridPane.rowIndex="12">
212                <GridPane.margin>
213                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
214                </GridPane.margin>
215             </Separator>
216             <Button fx:id="saveBtn" mnemonicParsing="false" style="-fx-
    border-color: a31f34;" text="Save Results" GridPane.columnSpan="
    2147483647" GridPane.halignment="CENTER" GridPane.rowIndex="13" />
217          </children>
218       </GridPane>
219    </right>
220 </fx:root>
```

A.1.6 UserInterfaceMonteCarlo.java

```
1
2  import javafx.application.Application;
3  import javafx.fxml.FXML;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.scene.control.SplitPane;
7  import javafx.stage.Stage;
8
9  public class UserInterfaceMonteCarlo extends Application {
10
11  @FXML
12  private Button resetBtn;
13
14  public static void main(String[] args) {
15   launch(args);
16  }
17
18  @Override
```

```
19  public void start(Stage primaryStage) throws Exception {
20
21   var mc_simulation = new DiversityMonteCarloSim(1);
22
23   UserInterfaceInputViewMonteCarlo iv = new
      UserInterfaceInputViewMonteCarlo();
24   UserInterfaceResultsViewMonteCarlo rv = new
      UserInterfaceResultsViewMonteCarlo();
25
26   iv.mc_simulation.set(mc_simulation);
27   rv.mc_simulation.set(mc_simulation);
28
29   iv.sessionProperty().bindBidirectional(rv.sessionProperty());
30
31   SplitPane sp = new SplitPane();
32
33   sp.getItems().addAll(iv, rv);
34
35   Scene scene = new Scene(sp);
36   primaryStage.setScene(scene);
37   primaryStage.show();
38
39  }
40 }
```

A.1.7 UserInterfaceInputViewMonteCarlo.java

```
1
2  import java.io.IOException;
3
4  import javafx.beans.property.ObjectProperty;
5  import javafx.beans.property.SimpleObjectProperty;
6  import javafx.fxml.FXML;
7  import javafx.fxml.FXMLLoader;
8  import javafx.scene.control.Button;
9  import javafx.scene.control.Label;
10 import javafx.scene.control.ProgressBar;
11 import javafx.scene.control.RadioButton;
12 import javafx.scene.control.TextField;
13 import javafx.scene.control.Tooltip;
14 import javafx.scene.layout.BorderPane;
15 import javafx.scene.layout.GridPane;
16 import javafx.util.converter.NumberStringConverter;
17
18 public class UserInterfaceInputViewMonteCarlo extends BorderPane {
19
20  @FXML
21  private GridPane inputPane;
22
23  @FXML
24  private TextField projectSpanTF, numLeadersTF, numEngineersTF,
      numSimulationsTF;
25
26  @FXML
```

```
27 private RadioButton EqualRadioBtn, DiffRadioBtn, tflOnBtn, tflOffBtn,
     sbaOnBtn, sbaOffBtn;
28
29 private DiversityApproach divApproach;
30
31 @FXML
32 private Button startBtn, continueBtn, resetBtn;
33
34 @FXML
35 private Label divApproachLabel, msgLabel;
36
37 @FXML
38 private ProgressBar progressBar;
39
40 public ObjectProperty<DiversityMonteCarloSim> mc_simulation;
41
42 public UserInterfaceInputViewMonteCarlo() {
43  FXMLLoader loader = new FXMLLoader(getClass().getResource("HistoInputs
      .fxml"));
44  loader.setRoot(this);
45  loader.setController(this);
46  try {
47   loader.load();
48  } catch (IOException e1) {
49   e1.printStackTrace();
50  }
51
52  mc_simulation = new SimpleObjectProperty<DiversityMonteCarloSim>();
53
54  mc_simulation.addListener((obs, oldV, newV) -> {
55   if (oldV != null) {
56    projectSpanTF.textProperty().unbindBidirectional(mc_simulation.get()
     .projectSpan);
57    numLeadersTF.textProperty().unbindBidirectional(mc_simulation.get().
     numLeaders);
58    numEngineersTF.textProperty().unbindBidirectional(mc_simulation.get
     ().numEngineers);
59    numSimulationsTF.textProperty().unbindBidirectional(mc_simulation.
     get().numSims);
60
61    tflOnBtn.selectedProperty().unbindBidirectional(mc_simulation.get().
     transformationalLeader);
62    tflOffBtn.selectedProperty().unbindBidirectional(mc_simulation.get()
     .transformationalLeader);
63
64    sbaOnBtn.selectedProperty().unbindBidirectional(mc_simulation.get().
     sociometricBadgeAwareness);
65    sbaOffBtn.selectedProperty().unbindBidirectional(mc_simulation.get()
     .sociometricBadgeAwareness);
66   }
67   if (newV != null) {
68    NumberStringConverter cv = new NumberStringConverter();
69    projectSpanTF.textProperty().bindBidirectional(mc_simulation.get().
     projectSpan, cv);
```

```
70      numLeadersTF.textProperty().bindBidirectional(mc_simulation.get().
        numLeaders, cv);
71      numEngineersTF.textProperty().bindBidirectional(mc_simulation.get().
        numEngineers, cv);
72      numSimulationsTF.textProperty().bindBidirectional(mc_simulation.get
        ().numSims, cv);
73
74
75      if (mc_simulation.get().transformationalLeader.get()) {
76       tflOnBtn.selectedProperty().bindBidirectional(mc_simulation.get().
        transformationalLeader);
77      } else {
78       tflOffBtn.selectedProperty().bindBidirectional(mc_simulation.get().
        transformationalLeader);
79      }
80
81      if (mc_simulation.get().sociometricBadgeAwareness.get()) {
82       sbaOnBtn.selectedProperty().bindBidirectional(mc_simulation.get().
        sociometricBadgeAwareness);
83      } else {
84       sbaOffBtn.selectedProperty().bindBidirectional(mc_simulation.get().
        sociometricBadgeAwareness);
85      }
86
87     }
88    });
89
90    EqualRadioBtn.setOnAction(e -> {
91
92     mc_simulation.get().setDiversityApproach(DiversityApproach.EQUALITY);
93     DiffRadioBtn.setSelected(false);
94    });
95
96    DiffRadioBtn.setOnAction(e -> {
97     mc_simulation.get().setDiversityApproach(DiversityApproach.DIFFERENCE
        );
98     EqualRadioBtn.setSelected(false);
99    });
100
101   tflOnBtn.setOnAction(e -> {
102    if (tflOnBtn.isSelected()) {
103     mc_simulation.get().setTransformationalLeader(true);
104     tflOffBtn.setSelected(false);
105     sbaOnBtn.setDisable(false);
106    } else {
107     mc_simulation.get().setTransformationalLeader(false);
108     tflOnBtn.setSelected(true);
109    }
110   });
111
112   tflOffBtn.setOnAction(e -> {
113    if (tflOffBtn.isSelected()) {
114     mc_simulation.get().setTransformationalLeader(false);
115     mc_simulation.get().setSociometricBadgeAwareness(false);
```

```
116     sbaOnBtn.setDisable(true);
117     sbaOffBtn.setSelected(true);
118     tflOnBtn.setSelected(false);
119    } else {
120     mc_simulation.get().setTransformationalLeader(true);
121     tflOffBtn.setSelected(true);
122    }
123   });
124
125   sbaOnBtn.setOnAction(e -> {
126    if (sbaOnBtn.isSelected()) {
127     mc_simulation.get().setSociometricBadgeAwareness(true);
128     sbaOffBtn.setSelected(false);
129    } else {
130     mc_simulation.get().setSociometricBadgeAwareness(false);
131     sbaOnBtn.setSelected(true);
132    }
133   });
134
135   sbaOffBtn.setOnAction(e -> {
136    if (sbaOffBtn.isSelected()) {
137     mc_simulation.get().setSociometricBadgeAwareness(false);
138     sbaOnBtn.setSelected(false);
139    } else {
140     mc_simulation.get().setSociometricBadgeAwareness(true);
141     sbaOffBtn.setSelected(true);
142    }
143   });
144
145   startBtn.setOnAction(e -> {
146    mc_simulation.get().start();
147    msgLabel.textProperty().set("Simulation in progress");
148    startBtn.setDisable(true);
149    continueBtn.setDisable(false);
150    continueBtn.setDefaultButton(true);
151    msgLabel.textProperty().set("Simulation Complete");
152
153   });
154
155   continueBtn.setOnAction(e -> {
156    mc_simulation.get().agentEnvironment.continueSim();
157   });
158
159   // brute force method to stop sim when demands are complete
160
161   resetBtn.setOnAction(e -> {
162    mc_simulation.get().resetSim();
163    // this.session = new SimpleObjectProperty<DiversityModelRunnerFX>();
164    // mc_simulation.get().agentEnvironment.setGlobalTime(0);
165    startBtn.setDisable(false);
166    startBtn.setDefaultButton(true);
167    continueBtn.setDisable(true);
168    msgLabel.textProperty().set(" ");
169   });
```

```
170
171   // Tooltips
172   Tooltip dvTooltip = new Tooltip();
173   dvTooltip.setText("Value in Equality vs. Value in Difference.  \n"
174     + "Value in Equality tend to give everyone equal opportunity, \n"
175     + "whereas Value in Difference approaches give someone a \n"
176     + "unique opportunity because of the skill sets he or she brings to
       the table.\n");
177   dvTooltip.install(divApproachLabel, dvTooltip);
178
179  }
180
181  // Getters and setters
182  public final ObjectProperty<DiversityMonteCarloSim> sessionProperty() {
183   return this.mc_simulation;
184  }
185
186  public final DiversityMonteCarloSim getSession() {
187   return this.sessionProperty().get();
188  }
189
190  public final void setSession(final DiversityMonteCarloSim session) {
191   this.sessionProperty().set(session);
192  }
193
194  public DiversityApproach getDivApproach() {
195   return divApproach;
196  }
197
198  public void setDivApproach(DiversityApproach divApproach) {
199   this.divApproach = divApproach;
200  }
201
202 }
```

A.1.8 UserInterfaceResultsViewMonteCarlo.java

```
1
2  import java.io.File;
3  import java.io.IOException;
4  import java.time.LocalDateTime;
5  import java.time.format.DateTimeFormatter;
6  import java.util.ArrayList;
7  import java.util.Collections;
8  import java.util.List;
9
10 import javafx.beans.property.ListProperty;
11 import javafx.beans.property.ObjectProperty;
12 import javafx.beans.property.SimpleListProperty;
13 import javafx.beans.property.SimpleObjectProperty;
14 import javafx.collections.FXCollections;
15 import javafx.collections.ListChangeListener;
16 import javafx.fxml.FXML;
17 import javafx.fxml.FXMLLoader;
```

```
18 import javafx.scene.chart.NumberAxis;
19 import javafx.scene.chart.ScatterChart;
20 import javafx.scene.chart.XYChart;
21 import javafx.scene.chart.XYChart.Data;
22 import javafx.scene.chart.XYChart.Series;
23 import javafx.scene.control.Button;
24 import javafx.scene.control.ChoiceBox;
25 import javafx.scene.control.TextField;
26 import javafx.scene.control.Tooltip;
27 import javafx.scene.layout.BorderPane;
28 import javafx.stage.FileChooser;
29 import javafx.util.converter.NumberStringConverter;
30
31 public class UserInterfaceResultsViewMonteCarlo extends BorderPane {
32
33  @FXML
34  private ScatterChart<Number, Number> dataSeriesScatter,
      sortedSeriesXScatter, sortedSeriesYScatter;
35
36  @FXML
37  private NumberAxis xAxisLabel, yAxisLabel;
38
39  // UI Properties
40  @FXML
41  private NumberAxis xAxisRetention, yAxisRetention;
42
43  @FXML
44  private NumberAxis xAxisDemands, yAxisDemands;
45
46  @FXML
47  private NumberAxis xAxisCommitment, yAxisEngagement;
48  @FXML
49  private TextField globalTimeTF;
50  @FXML
51  private TextField globalDemandsMootTF, globalDiversityDemandsMootTF;
52  @FXML
53  private TextField agentsLostLowTF, agentsLostModTF, agentsLostHighTF;
54  @FXML
55  private TextField indStartTF, indFinishTF;
56  @FXML
57  private Button saveBtn;
58
59  @FXML
60  private ChoiceBox<Object> xAxisChoiceBox, yAxisChoiceBox;//
      yAxis2ChoiceBox;
61
62  private int numRuns;
63
64  // Collections of results
65  private ListProperty<Data<Number, Number>> indStart;
66  private ListChangeListener<Data<Number, Number>> indStartListener;
67  private XYChart.Series<Number, Number> indStartSeries;
68
69  private ListProperty<Data<Number, Number>> indFinal;
```

```
70  private ListChangeListener<Data<Number, Number>> indFinalListener;
71  private XYChart.Series<Number, Number> indFinalSeries;
72
73  private ListProperty<Data<Number, Number>> projectDuration;
74  private ListChangeListener<Data<Number, Number>>
      projectDurationListener;
75  private XYChart.Series<Number, Number> projectDurationSeries;
76
77  private ListProperty<Data<Number, Number>> numDiversityConcernsTotal;
78  private ListChangeListener<Data<Number, Number>>
      numDiversityConcernsTotalListener;
79  private XYChart.Series<Number, Number> numDiversityConcernsTotalSeries;
80
81  private ListProperty<Data<Number, Number>> numDiversityConcernsComplete
      ;
82  private ListChangeListener<Data<Number, Number>>
      numDiversityConcernsCompleteListener;
83  private XYChart.Series<Number, Number>
      numDiversityConcernsCompleteSeries;
84
85  private ListProperty<Data<Number, Number>> numInteractions;
86  private ListChangeListener<Data<Number, Number>>
      numInteractionsListener;
87  private XYChart.Series<Number, Number> numInteractionsSeries;
88
89  private ListProperty<Data<Number, Number>> leaderInterventions;
90  private ListChangeListener<Data<Number, Number>>
      leaderInterventionsListener;
91  private XYChart.Series<Number, Number> leaderInterventionsSeries;
92
93  private ListProperty<Data<Number, Number>> leaderMicroSupports;
94  private ListChangeListener<Data<Number, Number>>
      leaderMicroSupportsListener;
95  private XYChart.Series<Number, Number> leaderMicroSupportsSeries;
96
97  private ListProperty<Data<Number, Number>> agentAbsences;
98  private ListChangeListener<Data<Number, Number>> agentAbsencesListener;
99  private XYChart.Series<Number, Number> agentAbsencesSeries;
100
101 private ListProperty<Data<Number, Number>> numAgentsLostLow;
102 private ListChangeListener<Data<Number, Number>>
      numAgentsLostLowListener;
103 private XYChart.Series<Number, Number> numAgentsLostLowSeries;
104
105 private ListProperty<Data<Number, Number>> numAgentsLostMod;
106 private ListChangeListener<Data<Number, Number>>
      numAgentsLostModListener;
107 private XYChart.Series<Number, Number> numAgentsLostModSeries;
108
109 private ListProperty<Data<Number, Number>> numAgentsLostHigh;
110 private ListChangeListener<Data<Number, Number>>
      numAgentsLostHighListener;
111 private XYChart.Series<Number, Number> numAgentsLostHighSeries;
112
```

```
113 // simulator & collected results
114 public SimpleObjectProperty<DiversityMonteCarloSim> mc_simulation;
115
116 // Constructor
117 public UserInterfaceResultsViewMonteCarlo() {
118  FXMLLoader loader = new FXMLLoader(getClass().getResource("
      MonteCarloResultsView.fxml"));
119  loader.setRoot(this);
120  loader.setController(this);
121  try {
122   loader.load();
123  } catch (IOException e1) {
124   e1.printStackTrace();
125  }
126
127  mc_simulation = new SimpleObjectProperty<DiversityMonteCarloSim>();
128  // numRuns = mc_simulation.get().numSims.get();
129
130  // Set up Choice Boxes
131  List<String> names = List.of("Interactions", "Starting IND", "Final
      IND", "Diversity Concerns",
132    "Agent Absences",
133    "Leader Interventions", "Total Leader Daily Supports", "Agents Lost
      (Low)", "Agents Lost (Moderate)",
134    "Agents Lost (High)", "Project Duration");
135
136  xAxisChoiceBox.getItems().addAll(names);
137  // xAxisChoiceBox.getSelectionModel().selectFirst();
138
139  yAxisChoiceBox.getItems().addAll(names);
140  // yAxis2ChoiceBox.getItems().addAll(names);
141  // yAxisChoiceBox.getSelectionModel().selectFirst();
142
143  // Establish listeners for sim environment results
144  // Listen for when an agent goes dormant
145
146  indStartListener = new ListChangeListener<Data<Number, Number>>() {
147   @Override
148   public void onChanged(javafx.collections.ListChangeListener.Change<?
      extends Data<Number, Number>> c) {
149
150    while (c.next()) {
151     if (c.wasAdded()) {
152      List<? extends Data<Number, Number>> addedSubList = c.
      getAddedSubList();
153       for (Data<Number, Number> x : addedSubList) {
154        addResultToChart(indStartSeries, x);
155       }
156     } else if (c.wasRemoved()) {
157      indStartSeries.getData().clear();
158      break;
159     }
160    }
161   }
```

```
162   };
163
164   indFinalListener = new ListChangeListener<Data<Number, Number>>() {
165    @Override
166    public void onChanged(javafx.collections.ListChangeListener.Change<?
       extends Data<Number, Number>> c) {
167
168     while (c.next()) {
169      if (c.wasAdded()) {
170       List<? extends Data<Number, Number>> addedSubList = c.
       getAddedSubList();
171       for (Data<Number, Number> x : addedSubList) {
172        addResultToChart(indFinalSeries, x);
173       }
174      } else if (c.wasRemoved()) {
175       indFinalSeries.getData().clear();
176       break;
177      }
178     }
179    }
180   };
181
182   projectDurationListener = new ListChangeListener<Data<Number, Number
       >>() {
183    @Override
184    public void onChanged(javafx.collections.ListChangeListener.Change<?
       extends Data<Number, Number>> c) {
185
186     while (c.next()) {
187      if (c.wasAdded()) {
188       List<? extends Data<Number, Number>> addedSubList = c.
       getAddedSubList();
189       for (Data<Number, Number> x : addedSubList) {
190        addResultToChart(projectDurationSeries, x);
191       }
192      } else if (c.wasRemoved()) {
193       projectDurationSeries.getData().clear();
194       break;
195      }
196     }
197    }
198   };
199
200   numDiversityConcernsTotalListener = new ListChangeListener<Data<Number
       , Number>>() {
201    @Override
202    public void onChanged(javafx.collections.ListChangeListener.Change<?
       extends Data<Number, Number>> c) {
203
204     while (c.next()) {
205      if (c.wasAdded()) {
206       List<? extends Data<Number, Number>> addedSubList = c.
       getAddedSubList();
207       for (Data<Number, Number> x : addedSubList) {
```

```
208         addResultToChart(numDiversityConcernsTotalSeries, x);
209        }
210       } else if (c.wasRemoved()) {
211        numDiversityConcernsTotalSeries.getData().clear();
212        break;
213       }
214      }
215     }
216    };
217
218    numDiversityConcernsCompleteListener = new ListChangeListener<Data<
       Number, Number>>() {
219     @Override
220     public void onChanged(javafx.collections.ListChangeListener.Change<?
       extends Data<Number, Number>> c) {
221
222      while (c.next()) {
223       if (c.wasAdded()) {
224        List<? extends Data<Number, Number>> addedSubList = c.
       getAddedSubList();
225        for (Data<Number, Number> x : addedSubList) {
226         addResultToChart(numDiversityConcernsCompleteSeries, x);
227        }
228       } else if (c.wasRemoved()) {
229        numDiversityConcernsCompleteSeries.getData().clear();
230        break;
231       }
232      }
233     }
234    };
235
236    numInteractionsListener = new ListChangeListener<Data<Number, Number
       >>() {
237     @Override
238     public void onChanged(javafx.collections.ListChangeListener.Change<?
       extends Data<Number, Number>> c) {
239
240      while (c.next()) {
241       if (c.wasAdded()) {
242        List<? extends Data<Number, Number>> addedSubList = c.
       getAddedSubList();
243        for (Data<Number, Number> x : addedSubList) {
244         addResultToChart(numInteractionsSeries, x);
245        }
246       } else if (c.wasRemoved()) {
247        numInteractionsSeries.getData().clear();
248        break;
249       }
250      }
251     }
252    };
253
254    agentAbsencesListener = new ListChangeListener<Data<Number, Number>>()
       {
```

```
255     @Override
256     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
257
258      while (c.next()) {
259       if (c.wasAdded()) {
260        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
261        for (Data<Number, Number> x : addedSubList) {
262         addResultToChart(agentAbsencesSeries, x);
263        }
264       } else if (c.wasRemoved()) {
265        agentAbsencesSeries.getData().clear();
266        break;
267       }
268      }
269     }
270    };
271
272    leaderInterventionsListener = new ListChangeListener<Data<Number,
        Number>>() {
273     @Override
274     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
275
276      while (c.next()) {
277       if (c.wasAdded()) {
278        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
279        for (Data<Number, Number> x : addedSubList) {
280         addResultToChart(leaderInterventionsSeries, x);
281        }
282       } else if (c.wasRemoved()) {
283        leaderInterventionsSeries.getData().clear();
284        break;
285       }
286      }
287     }
288    };
289
290    leaderMicroSupportsListener = new ListChangeListener<Data<Number,
        Number>>() {
291     @Override
292     public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
293
294      while (c.next()) {
295       if (c.wasAdded()) {
296        List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
297        for (Data<Number, Number> x : addedSubList) {
298         addResultToChart(leaderMicroSupportsSeries, x);
299        }
300       } else if (c.wasRemoved()) {
```

```
301         leaderMicroSupportsSeries.getData().clear();
302         break;
303        }
304       }
305      }
306     };
307
308     numAgentsLostLowListener = new ListChangeListener<Data<Number, Number
        >>() {
309      @Override
310      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
311
312       while (c.next()) {
313        if (c.wasAdded()) {
314         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
315         for (Data<Number, Number> x : addedSubList) {
316          addResultToChart(numAgentsLostLowSeries, x);
317         }
318        } else if (c.wasRemoved()) {
319         numAgentsLostLowSeries.getData().clear();
320         break;
321        }
322       }
323      }
324     };
325
326     numAgentsLostModListener = new ListChangeListener<Data<Number, Number
        >>() {
327      @Override
328      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
329
330       while (c.next()) {
331        if (c.wasAdded()) {
332         List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
333         for (Data<Number, Number> x : addedSubList) {
334          addResultToChart(numAgentsLostModSeries, x);
335         }
336        } else if (c.wasRemoved()) {
337         numAgentsLostModSeries.getData().clear();
338         break;
339        }
340       }
341      }
342     };
343
344     numAgentsLostHighListener = new ListChangeListener<Data<Number, Number
        >>() {
345      @Override
346      public void onChanged(javafx.collections.ListChangeListener.Change<?
        extends Data<Number, Number>> c) {
```

```
347
348     while (c.next()) {
349      if (c.wasAdded()) {
350       List<? extends Data<Number, Number>> addedSubList = c.
        getAddedSubList();
351       for (Data<Number, Number> x : addedSubList) {
352        addResultToChart(numAgentsLostHighSeries, x);
353       }
354      } else if (c.wasRemoved()) {
355       numAgentsLostHighSeries.getData().clear();
356       break;
357      }
358     }
359    }
360   };
361
362   // Initialize lists for storing results and add binding listener to
       them
363
364   indStart = new SimpleListProperty<Data<Number, Number>>(FXCollections.
       observableArrayList());
365   indStart.addListener(indStartListener);
366
367   indFinal = new SimpleListProperty<Data<Number, Number>>(FXCollections.
       observableArrayList());
368   indFinal.addListener(indFinalListener);
369
370   projectDuration = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
371   projectDuration.addListener(projectDurationListener);
372
373   numDiversityConcernsTotal = new SimpleListProperty<Data<Number, Number
       >>(FXCollections.observableArrayList());
374   numDiversityConcernsTotal.addListener(
       numDiversityConcernsTotalListener);
375
376   numDiversityConcernsComplete = new SimpleListProperty<Data<Number,
       Number>>(
377     FXCollections.observableArrayList());
378   numDiversityConcernsComplete.addListener(
       numDiversityConcernsCompleteListener);
379
380   numInteractions = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
381   numInteractions.addListener(numInteractionsListener);
382
383   agentAbsences = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
384   agentAbsences.addListener(agentAbsencesListener);
385
386   leaderInterventions = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
387   leaderInterventions.addListener(leaderInterventionsListener);
388
```

```
389   leaderMicroSupports = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
390   leaderMicroSupports.addListener(leaderMicroSupportsListener);
391
392   numAgentsLostLow = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
393   numAgentsLostLow.addListener(numAgentsLostLowListener);
394
395   numAgentsLostMod = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
396   numAgentsLostMod.addListener(numAgentsLostModListener);
397
398   numAgentsLostHigh = new SimpleListProperty<Data<Number, Number>>(
       FXCollections.observableArrayList());
399   numAgentsLostHigh.addListener(numAgentsLostHighListener);
400
401   // Establish a listener for the simulation to update User Interface
402   mc_simulation.addListener((obs, oldV, newV) -> {
403
404    if (oldV != null) {
405
406     globalTimeTF.textProperty().unbind();
407     globalTimeTF.textProperty().unbindBidirectional(mc_simulation.get().
       agentEnvironment.globalTime);
408
409     globalDemandsMootTF.textProperty()
410       .unbindBidirectional(mc_simulation.get().agentEnvironment.
       n_completeDemands);
411
412    }
413
414    if (newV != null) {
415     NumberStringConverter cv = new NumberStringConverter();
416
417     // Update data for charts
418     indStart.bind(mc_simulation.get().indStart_seriesProperty());
419     indFinal.bind(mc_simulation.get().indFinal_seriesProperty());
420
421     projectDuration.bind(mc_simulation.get().
       projectDuration_dataSeriesProperty());
422     numDiversityConcernsTotal.bind(mc_simulation.get().
       numDiversityDemandsTotal_dataSeriesProperty());
423     numDiversityConcernsComplete.bind(mc_simulation.get().
       numDiversityDemandsComplete_dataSeriesProperty());
424     numInteractions.bind(mc_simulation.get().
       numInteractions_dataSeriesProperty());
425     leaderInterventions.bind(mc_simulation.get().
       numLeaderInterventions_dataSeriesProperty());
426     leaderMicroSupports.bind(mc_simulation.get().
       numLeaderMicroSupports_dataSeriesProperty());
427     agentAbsences.bind(mc_simulation.get().
       numAgentAbsences_dataSeriesProperty());
428     numAgentsLostLow.bind(mc_simulation.get().
       numAgentsLostLow_dataSeriesProperty());
```

```
429     numAgentsLostMod.bind(mc_simulation.get().
        numAgentsLostMod_dataSeriesProperty());
430     numAgentsLostHigh.bind(mc_simulation.get().
        numAgentsLostHigh_dataSeriesProperty());
431 //
432 //    // Update text fields
433 //    globalTimeTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.globalTimeDays, cv);
434 //    globalDemandsMootTF.textProperty()
435 //      .bindBidirectional(session.get().agentEnvironment.
        n_completeDemands, cv);
436 //    globalDiversityDemandsMootTF.textProperty()
437 //      .bindBidirectional(session.get().agentEnvironment.
        n_completeDiversityDemands, cv);
438 //    agentsLostLowTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_low, cv);
439 //    agentsLostModTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_moderate,
440 //      cv);
441 //    agentsLostHighTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.n_agentsLost_high, cv);
442 //    indStartTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.indStart, cv);
443 //    indFinishTF.textProperty().bindBidirectional(session.get().
        agentEnvironment.indFinish, cv);
444     }
445   });
446
447   // Add listeners to the axis choiceboxes
448   xAxisChoiceBox.getSelectionModel().selectedItemProperty().addListener
        ((obs, oldV, newV) -> {
449    axesChanged();
450   });
451   yAxisChoiceBox.getSelectionModel().selectedItemProperty().addListener
        ((obs, oldV, newV) -> {
452    axesChanged();
453   });
454   // yAxis2ChoiceBox.getSelectionModel().selectedItemProperty().
        addListener((obs,
455   // oldV, newV) -> {
456 //    axes2Changed();
457 //  });
458
459   // Initialize charts & add data to chart
460   // Internal Network Density History
461   indStartSeries = new XYChart.Series<Number, Number>();
462   indStartSeries.setName("Starting Internal Network Density");
463
464   indFinalSeries = new XYChart.Series<Number, Number>();
465   indFinalSeries.setName("Final Internal Network Density");
466   // tradespaceScatter.getData().add(indDeltaSeries);
467
468   projectDurationSeries = new XYChart.Series<Number, Number>();
469   projectDurationSeries.setName("Project Duration");
```

```
470
471   numDiversityConcernsTotalSeries = new XYChart.Series<Number, Number>()
        ;
472   numDiversityConcernsTotalSeries.setName("Diversity Concerns");
473
474   numDiversityConcernsCompleteSeries = new XYChart.Series<Number, Number
        >();
475   numDiversityConcernsCompleteSeries.setName("Diversity Demands Complete
        ");
476
477   numInteractionsSeries = new XYChart.Series<Number, Number>();
478   numInteractionsSeries.setName("Agent Interactions");
479
480   agentAbsencesSeries = new XYChart.Series<Number, Number>();
481   agentAbsencesSeries.setName("Agent Absences");
482
483   leaderInterventionsSeries = new XYChart.Series<Number, Number>();
484   leaderInterventionsSeries.setName("Leader Interventions");
485
486   leaderMicroSupportsSeries = new XYChart.Series<Number, Number>();
487   leaderMicroSupportsSeries.setName("Total Leader Daily Supports");
488
489   numAgentsLostLowSeries = new XYChart.Series<Number, Number>();
490   numAgentsLostLowSeries.setName("Low-rep Agents Lost");
491
492   numAgentsLostModSeries = new XYChart.Series<Number, Number>();
493   numAgentsLostModSeries.setName("Mod-rep Agents Lost");
494
495   numAgentsLostHighSeries = new XYChart.Series<Number, Number>();
496   numAgentsLostHighSeries.setName("High-rep Agents Lost");
497
498
499   saveBtn.setOnAction(e -> {
500    saveResults();
501   });
502  }
503
504  public final ObjectProperty<DiversityMonteCarloSim> sessionProperty() {
505   return this.mc_simulation;
506  }
507
508  public final DiversityMonteCarloSim getSession() {
509   return this.sessionProperty().get();
510  }
511
512  public final void setSession(final DiversityMonteCarloSim session) {
513   this.sessionProperty().set(session);
514  }
515
516  private void addResultToChart(Series<Number, Number> dataSeries, Data<
       Number, Number> x) {
517   var d = new XYChart.Data<Number, Number>(x.getXValue(), x.getYValue())
        ;
518   d.setExtraValue(x);
```

```
519   dataSeries.getData().add(d);
520   Tooltip tooltip = new Tooltip();
521   tooltip.setText("" + x.getXValue() + ", " + x.getYValue());
522   Tooltip.install(d.getNode(), tooltip);
523  }
524
525  public void axesChanged() {
526   Object xAxis = xAxisChoiceBox.getSelectionModel().getSelectedItem();
527   Object yAxis = yAxisChoiceBox.getSelectionModel().getSelectedItem();
528
529   setAxesScatterXY(xAxis, yAxis);
530  }
531
532 // public void axes2Changed() {
533 //  Object yAxis = yAxis2ChoiceBox.getSelectionModel().getSelectedItem()
      ;
534 //
535 //  setAxes2ScatterXY(yAxis);
536 // }
537
538  public void setAxesScatterXY(Object x, Object y) {
539 //  "Project Duration", "Final IND", "Diversity Demands Generated",
540 //  "Diversity Demands Complete", "Interactions", "Agent Absences", "
      Leader Interventions",
541 //  "Leader Daily Supports",
542 //  "Agents Lost (Low)", "Agents Lost (Moderate)", "Agents Lost (High)"
543
544   XYChart.Series<Number, Number> newSeries = new XYChart.Series<Number,
       Number>();
545   XYChart.Series<Number, Number> newData = new XYChart.Series<Number,
       Number>();
546
547   XYChart.Series<Number, Number> newSeries1 = new XYChart.Series<Number,
        Number>();
548   XYChart.Series<Number, Number> newSeries2 = new XYChart.Series<Number,
        Number>();
549
550   numRuns = mc_simulation.get().numSims.get();
551   dataSeriesScatter.getData().clear();
552   sortedSeriesXScatter.getData().clear();
553   sortedSeriesYScatter.getData().clear();
554
555   List sortedArrayX = new ArrayList();
556   List sortedArrayY = new ArrayList();
557
558
559   if (x == null || y == null) {
560    return;
561   }
562
563   ListProperty<Data<Number, Number>> newXdata;
564   ListProperty<Data<Number, Number>> newYdata;
565
566
```

```
567   Series<Number, Number> newXvalues = new Series<Number, Number>();
568   Series<Number, Number> newYvalues = new Series<Number, Number>();
569
570   if (x == "Project Duration") {
571    newXvalues = projectDurationSeries;
572   } else if (x == "Starting IND") {
573    newXvalues = indStartSeries;
574   } else if (x == "Final IND") {
575    newXvalues = indFinalSeries;
576   } else if (x == "Diversity Concerns") {
577    newXvalues = numDiversityConcernsTotalSeries;
578   } else if (x == "Diversity Demands Complete") {
579    newXvalues = numDiversityConcernsCompleteSeries;
580   } else if (x == "Interactions") {
581    newXvalues = numInteractionsSeries;
582   } else if (x == "Agent Absences") {
583    newXvalues = agentAbsencesSeries;
584   } else if (x == "Leader Interventions") {
585    newXvalues = leaderInterventionsSeries;
586   } else if (x == "Total Leader Daily Supports") {
587    newXvalues = leaderMicroSupportsSeries;
588   } else if (x == "Agents Lost (Low)") {
589    newXvalues = numAgentsLostLowSeries;
590   } else if (x == "Agents Lost (Moderate)") {
591    newXvalues = numAgentsLostModSeries;
592   } else if (x == "Agents Lost (High)") {
593    newXvalues = numAgentsLostHighSeries;
594   }
595
596   if (y == "Project Duration") {
597    newYvalues = projectDurationSeries;
598   } else if (y == "Starting IND") {
599    newYvalues = indStartSeries;
600   } else if (y == "Final IND") {
601    newYvalues = indFinalSeries;
602   } else if (y == "Diversity Concerns") {
603    newYvalues = numDiversityConcernsTotalSeries;
604   } else if (y == "Diversity Demands Complete") {
605    newYvalues = numDiversityConcernsCompleteSeries;
606   } else if (y == "Agent Absences") {
607    newYvalues = agentAbsencesSeries;
608   } else if (y == "Interactions") {
609    newYvalues = numInteractionsSeries;
610   } else if (y == "Leader Interventions") {
611    newYvalues = leaderInterventionsSeries;
612   } else if (y == "Total Leader Daily Supports") {
613    newYvalues = leaderMicroSupportsSeries;
614   } else if (y == "Agents Lost (Low)") {
615    newYvalues = numAgentsLostLowSeries;
616   } else if (y == "Agents Lost (Moderate)") {
617    newYvalues = numAgentsLostModSeries;
618   } else if (y == "Agents Lost (High)") {
619    newYvalues = numAgentsLostHighSeries;
620   }
```

```
621
622   if (newXvalues == null || newYvalues == null) {
623    return;
624   } else {
625
626    for (int i = 0; i < numRuns; i++) {
627     newSeries.getData().add(new Data<Number, Number>(newXvalues.getData
      ().get(i).getYValue(),
628       newYvalues.getData().get(i).getYValue()));
629     Tooltip tooltip = new Tooltip();
630     tooltip.setText(
631       "" + newXvalues.getData().get(i).getYValue() + ", " + newYvalues.
      getData().get(i).getYValue());
632     Tooltip.install(newSeries.getData().get(i).getNode(), tooltip);
633
634 //    if (agentAbsencesSeries.getData().get(i).getYValue().intValue() <=
       3) {
635 //      System.out.println("true");
636 //      newData.getData().add(new Data<Number, Number>(newXvalues.getData
      ().get(i).getYValue(),
637 //        newYvalues.getData().get(i).getYValue()));
638 //    }
639
640     sortedArrayX.add(newXvalues.getData().get(i).getYValue());
641     sortedArrayY.add(newYvalues.getData().get(i).getYValue());
642    }
643    // newData.getNode().setStyle("-fx-background-color: #002244,
      #002244");
644
645    Collections.sort(sortedArrayX);
646    Collections.sort(sortedArrayY);
647
648    for (int i = 0; i < numRuns; i++) {
649     newSeries1.getData().add(new Data<Number, Number>(i, (Number)
      sortedArrayX.get(i)));
650
651     newSeries2.getData().add(new Data<Number, Number>(i, (Number)
      sortedArrayY.get(i)));
652    }
653
654
655    dataSeriesScatter.getData().add(newSeries);
656 //    dataSeriesScatter.getData().add(newData);
657    dataSeriesScatter.setStyle("CHART_COLOR_2:" + 002244);
658    dataSeriesScatter.getXAxis().labelProperty().set(newXvalues.getName()
      );
659    dataSeriesScatter.getYAxis().labelProperty().set(newYvalues.getName()
      );
660    dataSeriesScatter.setTitle(newYvalues.getName() + " vs " + newXvalues
      .getName());
661
662    sortedSeriesXScatter.getData().addAll(newSeries1);
663    sortedSeriesXScatter.getXAxis().labelProperty().set("Sorted");
664    sortedSeriesXScatter.getYAxis().labelProperty().set(newXvalues.
```

```
    getName());
665   sortedSeriesXScatter.setTitle(newXvalues.getName());
666
667   sortedSeriesYScatter.getData().addAll(newSeries2);
668   sortedSeriesYScatter.getXAxis().labelProperty().set("Sorted");
669   sortedSeriesYScatter.getYAxis().labelProperty().set(newYvalues.
    getName());
670   sortedSeriesYScatter.setTitle(newYvalues.getName());
671
672  }
673 }
674
675 public void setAxes2ScatterXY(Object y) {
676 //  "Project Duration", "Starting IND", "Final IND", "Diversity Demands
    Generated",
677 //  "Diversity Demands Complete", "Interactions", "Agent Absences", "
    Leader Interventions",
678 //  "Total Leader Daily Supports",
679 //  "Agents Lost (Low)", "Agents Lost (Moderate)", "Agents Lost (High)"
680  XYChart.Series<Number, Number> newSeries = new XYChart.Series<Number,
    Number>();
681  numRuns = mc_simulation.get().numSims.get();
682  sortedSeriesXScatter.getData().clear();
683  if (y == null) {
684   return;
685  }
686
687  Series<Number, Number> newYvalues = new Series<Number, Number>();
688
689  if (y == "Project Duration") {
690   newYvalues = projectDurationSeries;
691  } else if (y == "Starting IND") {
692   newYvalues = indStartSeries;
693  } else if (y == "Final IND") {
694   newYvalues = indFinalSeries;
695  } else if (y == "Diversity Demands Generated") {
696   newYvalues = numDiversityConcernsTotalSeries;
697  } else if (y == "Diversity Demands Complete") {
698   newYvalues = numDiversityConcernsCompleteSeries;
699  } else if (y == "Agent Absences") {
700   newYvalues = agentAbsencesSeries;
701  } else if (y == "Interactions") {
702   newYvalues = numInteractionsSeries;
703  } else if (y == "Leader Interventions") {
704   newYvalues = leaderInterventionsSeries;
705  } else if (y == "Total ") {
706   newYvalues = leaderMicroSupportsSeries;
707  } else if (y == "Agents Lost (Low)") {
708   newYvalues = numAgentsLostLowSeries;
709  } else if (y == "Agents Lost (Moderate)") {
710   newYvalues = numAgentsLostModSeries;
711  } else if (y == "Agents Lost (High)") {
712   newYvalues = numAgentsLostHighSeries;
713  }
```

```
714
715   if (newYvalues == null) {
716
717    return;
718   } else {
719
720    List sortedArray = new ArrayList();
721    for (int i = 0; i < numRuns; i++) {
722     sortedArray.add(newYvalues.getData().get(i).getYValue());
723     Collections.sort(sortedArray);
724    }
725
726    for (int i = 0; i < numRuns; i++) {
727     newSeries.getData().add(new Data<Number, Number>(i, (Number)
      sortedArray.get(i)));
728    }
729
730    sortedSeriesXScatter.getData().add(newSeries);
731    sortedSeriesXScatter.getXAxis().labelProperty().set("Simulation #");
732    sortedSeriesXScatter.getYAxis().labelProperty().set(newYvalues.
      getName());
733   }
734  }
735
736  public void saveResults() {
737   DateTimeFormatter dtf = DateTimeFormatter.ofPattern(" yyyy-MM-
      dd_HHmmss");
738   LocalDateTime now = LocalDateTime.now();
739   var chooserOut = new FileChooser();
740   chooserOut.setTitle("Please save the results to a file");
741   chooserOut.setInitialFileName("DiversityABM_ResultData_" + dtf.format(
      now) + ".xlsx");
742   chooserOut.getExtensionFilters().addAll(new FileChooser.
      ExtensionFilter("Excel Workbook Files", "*.xlsx"));
743
744   File fileToWrite = chooserOut.showSaveDialog(null);
745
746   var re = new ResultsExporter(mc_simulation.get().simResults);
747   re.export(fileToWrite);
748   ResultsExporter results = new ResultsExporter(mc_simulation.get().
      simResults);
749  }
750
751 }
```

A.1.9 HistoInputs.fxml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.geometry.Insets?>
4 <?import javafx.scene.control.Button?>
5 <?import javafx.scene.control.Label?>
6 <?import javafx.scene.control.RadioButton?>
7 <?import javafx.scene.control.Separator?>
```

```
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.BorderPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.text.Font?>

<fx:root maxWidth="-Infinity" minWidth="-Infinity" prefWidth="450.0" type="BorderPane" xmlns="http://javafx.com/javafx/16" xmlns:fx="http://javafx.com/fxml/1">
   <top>
      <Label alignment="CENTER" minWidth="400.0" prefHeight="39.0" prefWidth="540.0" text="Simulation Inputs" BorderPane.alignment="CENTER">
         <font>
            <Font size="32.0" />
         </font>
      </Label>
   </top>
   <center>
      <GridPane fx:id="inputPane" BorderPane.alignment="CENTER">
        <columnConstraints>
          <ColumnConstraints halignment="RIGHT" hgrow="ALWAYS" />
          <ColumnConstraints halignment="CENTER" hgrow="ALWAYS" maxWidth="135.0" minWidth="125.66665649414062" prefWidth="135.0" />
          <ColumnConstraints halignment="CENTER" hgrow="ALWAYS" maxWidth="46.0" minWidth="0.0" prefWidth="1.0" />
          <ColumnConstraints halignment="CENTER" hgrow="ALWAYS" maxWidth="175.5" minWidth="100.5" prefWidth="175.5" />
          <ColumnConstraints />
        </columnConstraints>
        <rowConstraints>
          <RowConstraints />
          <RowConstraints />
          <RowConstraints maxHeight="17.0" minHeight="16.333332061767578" prefHeight="16.333332061767578" vgrow="SOMETIMES" />
          <RowConstraints maxHeight="17.666667938232422" minHeight="17.0" prefHeight="17.666667938232422" vgrow="SOMETIMES" />
          <RowConstraints maxHeight="27.5" minHeight="27.5" prefHeight="27.5" vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
         <RowConstraints percentHeight="-2.0" vgrow="SOMETIMES" />
          <RowConstraints maxHeight="30.5" minHeight="10.0" percentHeight="0.0" prefHeight="11.0" vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints vgrow="SOMETIMES" />
          <RowConstraints maxHeight="19.0" minHeight="7.0" prefHeight="19.0" vgrow="SOMETIMES" />
          <RowConstraints maxHeight="26.0" minHeight="10.0" prefHeight="14.0" vgrow="SOMETIMES" />
```

```
48          <RowConstraints maxHeight="26.0" minHeight="10.0" prefHeight
    ="14.0" vgrow="SOMETIMES" />
49          <RowConstraints maxHeight="52.0" minHeight="40.0"
    percentHeight="0.0" prefHeight="52.0" vgrow="SOMETIMES" />
50          <RowConstraints percentHeight="0.0" vgrow="SOMETIMES" />
51          <RowConstraints percentHeight="0.0" vgrow="SOMETIMES" />
52          <RowConstraints minHeight="10.0" percentHeight="0.0"
    prefHeight="30.0" vgrow="SOMETIMES" />
53       </rowConstraints>
54        <children>
55          <Label alignment="CENTER" contentDisplay="CENTER" text="
    Project Duration (days)" wrapText="true" GridPane.rowIndex="10" />
56          <TextField fx:id="projectSpanTF" maxWidth="-Infinity"
    prefWidth="90.0" style="-fx-border-color: a31f34;" GridPane.
    columnIndex="1" GridPane.rowIndex="10">
57             <GridPane.margin>
58                <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
59             </GridPane.margin>
60          </TextField>
61          <TextField fx:id="numLeadersTF" maxWidth="-Infinity"
    prefWidth="90.0" style="-fx-border-color: a31f34;" GridPane.
    columnIndex="1" GridPane.rowIndex="11">
62             <GridPane.margin>
63                <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
64             </GridPane.margin>
65          </TextField>
66          <TextField fx:id="numEngineersTF" maxWidth="-Infinity"
    prefWidth="90.0" style="-fx-border-color: a31f34;" GridPane.
    columnIndex="1" GridPane.rowIndex="12">
67             <GridPane.margin>
68                <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
69             </GridPane.margin>
70          </TextField>
71          <TextField fx:id="numSimulationsTF" maxWidth="-Infinity"
    prefWidth="90.0" style="-fx-border-color: a31f34;" GridPane.
    columnIndex="1" GridPane.rowIndex="14">
72             <GridPane.margin>
73                <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
74             </GridPane.margin>
75          </TextField>
76          <Label text="# of Formal Leaders" wrapText="true" GridPane.
    rowIndex="11" />
77          <Label text="# of Project Engineers" wrapText="true"
    GridPane.rowIndex="12" />
78          <Label fx:id="divApproachLabel" text="Diversity Value
    Approach" textAlignment="CENTER" wrapText="true" GridPane.columnSpan
    ="2" GridPane.halignment="CENTER" GridPane.rowIndex="2" />
79          <Button fx:id="startBtn" defaultButton="true" minWidth="-
    Infinity" mnemonicParsing="false" prefWidth="75.0" style="-fx-border
    -color: a31f34;" text="Start" GridPane.halignment="CENTER" GridPane.
```

```
rowIndex="16">
80                <GridPane.margin>
81                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
82                </GridPane.margin></Button>
83             <Button fx:id="continueBtn" disable="true" minWidth="-
Infinity" mnemonicParsing="false" prefWidth="75.0" style="-fx-border
-color: a31f34;" text="Continue" GridPane.columnIndex="1" GridPane.
columnSpan="2" GridPane.rowIndex="16">
84                <GridPane.margin>
85                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
86                </GridPane.margin></Button>
87             <Button fx:id="resetBtn" minWidth="-Infinity"
mnemonicParsing="false" prefWidth="75.0" style="-fx-border-color:
a31f34;" text="Reset" GridPane.columnIndex="3" GridPane.columnSpan="
2147483647" GridPane.halignment="CENTER" GridPane.rowIndex="16">
88                <GridPane.margin>
89                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
/>
90                </GridPane.margin></Button>
91             <RadioButton fx:id="EqualRadioBtn" minWidth="-Infinity"
mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
90.0" selected="true" text="Equality" GridPane.halignment="CENTER"
GridPane.rowIndex="4">
92                <GridPane.margin>
93                   <Insets bottom="10.0" left="10.0" right="10.0" />
94                </GridPane.margin></RadioButton>
95             <RadioButton fx:id="DiffRadioBtn" minWidth="-Infinity"
mnemonicParsing="false" prefWidth="90.0" text="Difference" GridPane.
columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="4">
96                <GridPane.margin>
97                   <Insets bottom="10.0" left="10.0" right="10.0" />
98                </GridPane.margin></RadioButton>
99             <RadioButton fx:id="tflOnBtn" minWidth="-Infinity"
mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
90.0" selected="true" text="On" GridPane.halignment="CENTER"
GridPane.rowIndex="6">
100               <GridPane.margin>
101                  <Insets bottom="10.0" left="10.0" right="10.0" />
102               </GridPane.margin>
103            </RadioButton>
104            <RadioButton fx:id="tflOffBtn" minWidth="-Infinity"
mnemonicParsing="false" prefWidth="90.0" text="Off" GridPane.
columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="6">
105               <GridPane.margin>
106                  <Insets bottom="10.0" left="10.0" right="10.0" />
107               </GridPane.margin>
108            </RadioButton>
109            <Label text="Transformational Leadership" textAlignment="
CENTER" wrapText="true" GridPane.columnSpan="2" GridPane.halignment=
"CENTER" GridPane.rowIndex="5" />
110            <Label text="Sociometric Badge Awareness" textAlignment="
CENTER" wrapText="true" GridPane.columnSpan="2" GridPane.halignment=
```

```
"CENTER" GridPane.rowIndex="7" />
111             <RadioButton fx:id="sbaOnBtn" minWidth="-Infinity"
    mnemonicParsing="false" nodeOrientation="RIGHT_TO_LEFT" prefWidth="
    90.0" selected="true" text="On" GridPane.halignment="CENTER"
    GridPane.rowIndex="8">
112                <GridPane.margin>
113                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
114                </GridPane.margin>
115             </RadioButton>
116             <RadioButton fx:id="sbaOffBtn" minWidth="-Infinity"
    mnemonicParsing="false" prefWidth="90.0" text="Off" GridPane.
    columnIndex="1" GridPane.halignment="CENTER" GridPane.rowIndex="8">
117                <GridPane.margin>
118                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
119                </GridPane.margin>
120             </RadioButton>
121             <Separator prefWidth="200.0" GridPane.columnSpan="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="9">
122                <GridPane.margin>
123                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
124                </GridPane.margin></Separator>
125             <Label text="# of Simulations" GridPane.rowIndex="14" />
126             <Label fx:id="msgLabel" GridPane.columnSpan="2147483647"
    GridPane.halignment="CENTER" GridPane.rowIndex="19" />
127             <Separator orientation="VERTICAL" prefHeight="200.0"
    GridPane.columnIndex="2" GridPane.halignment="RIGHT" GridPane.
    rowIndex="2" GridPane.rowSpan="11">
128                <GridPane.margin>
129                   <Insets />
130                </GridPane.margin>
131             </Separator>
132             <RadioButton fx:id="dvApproachRandomBtn" disable="true"
    mnemonicParsing="false" text="Randomize" GridPane.columnIndex="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="4">
133                <GridPane.margin>
134                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
135                </GridPane.margin></RadioButton>
136             <RadioButton fx:id="tflRandomBtn" disable="true"
    mnemonicParsing="false" text="Randomize" GridPane.columnIndex="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="6">
137                <GridPane.margin>
138                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
139                </GridPane.margin></RadioButton>
140             <RadioButton fx:id="sbaRandomBtn" disable="true"
    mnemonicParsing="false" text="Randomize" GridPane.columnIndex="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="8">
141                <GridPane.margin>
142                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
```

```
143                </GridPane.margin></RadioButton>
144             <RadioButton fx:id="pdRandomBtn" disable="true"
    mnemonicParsing="false" text="Randomize" GridPane.columnIndex="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="10">
145                <GridPane.margin>
146                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
147                </GridPane.margin></RadioButton>
148             <RadioButton disable="true" mnemonicParsing="false" text="
    Randomize" GridPane.columnIndex="3" GridPane.halignment="LEFT"
    GridPane.rowIndex="11">
149                <GridPane.margin>
150                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
151                </GridPane.margin></RadioButton>
152             <RadioButton fx:id="numEngineersRandomBtn" disable="true"
    mnemonicParsing="false" text="Randomize" GridPane.columnIndex="3"
    GridPane.halignment="LEFT" GridPane.rowIndex="12">
153                <GridPane.margin>
154                   <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
155                </GridPane.margin></RadioButton>
156             <Separator prefWidth="200.0" GridPane.columnSpan="2147483647
    " GridPane.rowIndex="1" />
157             <Separator prefWidth="200.0" GridPane.columnSpan="2147483647
    " GridPane.rowIndex="13" GridPane.valignment="TOP">
158                <GridPane.margin>
159                   <Insets bottom="5.0" left="5.0" right="5.0" top="5.0"
    />
160                </GridPane.margin></Separator>
161          </children>
162       </GridPane>
163    </center>
164 </fx:root>
```

A.1.10 MonteCarloResultsView.fxml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.geometry.Insets?>
4  <?import javafx.scene.chart.NumberAxis?>
5  <?import javafx.scene.chart.ScatterChart?>
6  <?import javafx.scene.control.Button?>
7  <?import javafx.scene.control.ChoiceBox?>
8  <?import javafx.scene.control.Label?>
9  <?import javafx.scene.control.SplitPane?>
10 <?import javafx.scene.effect.Blend?>
11 <?import javafx.scene.layout.BorderPane?>
12 <?import javafx.scene.layout.ColumnConstraints?>
13 <?import javafx.scene.layout.GridPane?>
14 <?import javafx.scene.layout.RowConstraints?>
15 <?import javafx.scene.text.Font?>
16
17 <fx:root prefWidth="1000.0" type="BorderPane" xmlns="http://javafx.com/
```

```
javafx/16" xmlns:fx="http://javafx.com/fxml/1">
<top>
   <Label text="Monte Carlo Simulation Results" BorderPane.alignment="CENTER">
      <font>
         <Font size="26.0" />
      </font>
   </Label>
</top>
<center>
   <GridPane maxWidth="1.7976931348623157E308" styleClass="application/application.css" BorderPane.alignment="CENTER">
      <columnConstraints>
         <ColumnConstraints hgrow="SOMETIMES" />
         <ColumnConstraints hgrow="SOMETIMES" />
         <ColumnConstraints hgrow="SOMETIMES" />
         <ColumnConstraints hgrow="SOMETIMES" maxWidth="1.7976931348623157E308" />
      </columnConstraints>
      <rowConstraints>
         <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
         <RowConstraints maxHeight="258.0" minHeight="10.0" prefHeight="258.0" vgrow="SOMETIMES" />
         <RowConstraints maxHeight="251.0" minHeight="10.0" prefHeight="46.0" vgrow="SOMETIMES" />
         <RowConstraints maxHeight="316.0" minHeight="0.0" prefHeight="174.0" vgrow="SOMETIMES" />
         <RowConstraints maxHeight="251.0" minHeight="10.0" prefHeight="172.0" vgrow="SOMETIMES" />
         <RowConstraints maxHeight="337.0" minHeight="47.0" prefHeight="47.0" vgrow="SOMETIMES" />
      </rowConstraints>
      <children>
         <SplitPane dividerPositions="0.5" orientation="VERTICAL" GridPane.columnIndex="1" GridPane.rowSpan="2147483647">
           <items>
               <GridPane>
                  <columnConstraints>
                     <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0" minWidth="10.0" prefWidth="320.0" />
                     <ColumnConstraints hgrow="SOMETIMES" maxWidth="496.0" minWidth="10.0" prefWidth="320.0" />
                     <ColumnConstraints hgrow="SOMETIMES" maxWidth="302.0" minWidth="10.0" prefWidth="280.0" />
                  </columnConstraints>
                  <rowConstraints>
                     <RowConstraints minHeight="10.0" vgrow="SOMETIMES" />
                  </rowConstraints>
                  <children>
                     <SplitPane dividerPositions="0.5" GridPane.columnSpan="2147483647">
                        <items>
```

```
56                              <ScatterChart fx:id="sortedSeriesYScatter"
 animated="false" prefWidth="350.0" style="-fx-border-color: a31f34;
" stylesheets="@application/application.css" title="Metric 1 (Sorted
)">
57                                <xAxis>
58                                  <NumberAxis side="BOTTOM" />
59                                </xAxis>
60                                <yAxis>
61                                  <NumberAxis side="LEFT" />
62                                </yAxis>
63                              </ScatterChart>
64                              <ScatterChart fx:id="sortedSeriesXScatter"
 animated="false" prefWidth="350.0" style="-fx-border-color: a31f34;
" stylesheets="@application/application.css" title="Metric 2 (Sorted
)">
65                                <xAxis>
66                                  <NumberAxis side="BOTTOM" />
67                                </xAxis>
68                                <yAxis>
69                                  <NumberAxis side="LEFT" />
70                                </yAxis>
71                              </ScatterChart>
72                            </items>
73                          </SplitPane>
74                        </children>
75                      </GridPane>
76                      <ScatterChart fx:id="dataSeriesScatter" animated="
false" prefWidth="600.0" style="-fx-border-color: a31f34;"
stylesheets="@application/application.css" title="Metric Comparison
(Metric 1 vs Metric 2)">
77                        <xAxis>
78                          <NumberAxis side="BOTTOM" fx:id="xAxisLabel" />
79                        </xAxis>
80                        <yAxis>
81                          <NumberAxis fx:id="yAxisLabel" side="LEFT" />
82                        </yAxis>
83                      </ScatterChart>
84                    </items>
85                  </SplitPane>
86                  <ChoiceBox minWidth="175.0" prefWidth="175.0" style="-fx-
border-color: a31f34;" GridPane.columnIndex="2" GridPane.halignment=
"LEFT" GridPane.rowIndex="3" GridPane.valignment="TOP" fx:id="
xAxisChoiceBox">
87                    <GridPane.margin>
88                      <Insets bottom="10.0" left="10.0" right="10.0" top="
10.0" />
89                    </GridPane.margin>
90                    <effect>
91                      <Blend opacity="0.0" />
92                    </effect></ChoiceBox>
93                  <Label accessibleHelp=")" text="Metric 2 (X Axis)" GridPane.
columnIndex="2" GridPane.halignment="CENTER" GridPane.rowIndex="2"
GridPane.valignment="BOTTOM">
94                    <GridPane.margin>
```

```
95                   <Insets left="10.0" right="10.0" />
96                </GridPane.margin></Label>
97             <Label prefHeight="0.0" prefWidth="88.0" text="Metric 1 (Y
    Axis)" GridPane.halignment="CENTER" GridPane.rowIndex="2" GridPane.
    valignment="BOTTOM">
98                <GridPane.margin>
99                   <Insets left="10.0" right="10.0" top="10.0" />
100               </GridPane.margin>
101            </Label>
102            <ChoiceBox fx:id="yAxisChoiceBox" minWidth="175.0" prefWidth
    ="175.0" style="-fx-border-color: a31f34;" GridPane.halignment="
    CENTER" GridPane.rowIndex="3" GridPane.valignment="TOP">
103               <GridPane.margin>
104                  <Insets bottom="10.0" left="10.0" right="10.0" top="
    10.0" />
105               </GridPane.margin>
106            </ChoiceBox>
107         </children>
108      </GridPane>
109   </center>
110   <bottom>
111      <Button fx:id="saveBtn" mnemonicParsing="false" style="-fx-border-
    color: a31f34;" text="Save Results" BorderPane.alignment="CENTER">
112         <BorderPane.margin>
113            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
114         </BorderPane.margin></Button>
115   </bottom>
116 </fx:root>
```

A.2 Environment Files

A.2.1 DiversityModelRunner.java

```
1
2  import java.util.Random;
3
4  import javafx.beans.property.BooleanProperty;
5  import javafx.beans.property.DoubleProperty;
6  import javafx.beans.property.IntegerProperty;
7  import javafx.beans.property.ListProperty;
8  import javafx.beans.property.ObjectProperty;
9  import javafx.beans.property.SimpleBooleanProperty;
10 import javafx.beans.property.SimpleDoubleProperty;
11 import javafx.beans.property.SimpleIntegerProperty;
12 import javafx.beans.property.SimpleListProperty;
13 import javafx.beans.property.SimpleObjectProperty;
14 import javafx.beans.property.SimpleStringProperty;
15 import javafx.beans.property.StringProperty;
16 import javafx.collections.FXCollections;
17 import javafx.collections.ObservableList;
18
19 /**
20  *
```

```
21  * @author Jeremy Goodwin with considerable support from Dr. Bryan Moser
        Written
22  *         in conjunction with thesis pursuant to MIT SDM degree, 2021
23  */
24
25 public class DiversityModelRunner {
26
27  // States (potentially) controlled by User Interface
28  IntegerProperty projectSpan;
29  IntegerProperty agentTimeStep;
30  IntegerProperty numLeaders;
31  IntegerProperty numEngineers;
32
33  ObjectProperty<DiversityApproach> diversityApproach;
34  BooleanProperty transformationalLeader;
35  BooleanProperty sociometricBadgeAwareness;
36
37  DoubleProperty progress;
38
39  ListProperty<Agent> agents;
40
41  ListProperty<DemandFX> initialDemandList;
42  World agentEnvironment;
43  SimResult simResults;
44
45  // Default Constructor
46  public DiversityModelRunner() {
47
48   // Create Initial Values for inclusion in UI
49   projectSpan = new SimpleIntegerProperty(28); // x-day project (TBR in
       UI)
50   numLeaders = new SimpleIntegerProperty(1); // x leader (TBR in UI)
51   numEngineers = new SimpleIntegerProperty(11); // x engineers (TBR in
       UI)
52   agentTimeStep = new SimpleIntegerProperty(1); // x day (TBR in UI)
53
54   // Leadership parameters from UI for agent
55   transformationalLeader = new SimpleBooleanProperty(true);
56   sociometricBadgeAwareness = new SimpleBooleanProperty(true);
57   diversityApproach = new SimpleObjectProperty<DiversityApproach>(
       DiversityApproach.EQUALITY);
58
59   // Create new "World" environment for agents to operate in
60   agentEnvironment = new World();
61   agentEnvironment.setDiversityApproach(diversityApproach);
62
63   // Set initial values for UI
64   agentEnvironment.setGlobalTime(0);
65
66   // Create blank list of globally-assigned demands
67   initialDemandList = new SimpleListProperty<DemandFX>(FXCollections.
       observableArrayList());
68
69  }
```

```
70
71  // Initialize the simulation
72  public void start() {
73
74   // Establish initial demands for engineers and leaders (not diversity
      related)
75   agentEnvironment
76     .setGlobalAssignedDemands(createInitialDesignDemands(projectSpan.get
      () * 8 * numEngineers.get()));
77   agentEnvironment.setGlobalAssignedDemands(createInitialLeaderDemands(
      projectSpan.get() * 8 * numLeaders.get()));
78
79   // Create initial set of agents based on UI input
80   agents = new SimpleListProperty<>(FXCollections.observableArrayList())
      ;
81   createEngineers(numEngineers.get());
82   createLeaders(numLeaders.get());
83
84   // Establish work time limits and get environment started
85   agents.stream().forEach(a -> agentEnvironment.addAgent(a));
86   agentEnvironment.agents.stream().forEach(a -> {
87    a.setTimeStep(new SimpleIntegerProperty(agentTimeStep.get() * 3600 *
      8));
88    a.setAbsenceThreshold(new SimpleDoubleProperty(0.10 * projectSpan.get
      ()));
89    a.setConsequenceThreshold(.10 * projectSpan.get());
90    a.setDiversityApproach(diversityApproach.get());
91   });
92   agentEnvironment.agents.stream().filter(a -> a.getType() == AgentType.
      LEADER).forEach(b -> {
93    b.setTransormationalLeader(transformationalLeader);
94    b.setSociometricBadgeAwareness(sociometricBadgeAwareness);
95   });
96
97   agentEnvironment.start();
98
99   // Create place to collect results
100  simResults = new SimResult(agentEnvironment);
101 }
102
103 // Convenience Methods
104 // Initial demands set
105 private ListProperty<DemandFX> createInitialDesignDemands(int
      numDemands) {
106  Random rand = new Random(); // allow for design or coord demands
107  int coordinationFactor = 20; // percentage of tasks requiring
      coordination
108
109  for (int d = 0; d < numDemands; d++) {
110   if (rand.nextInt(100) > (coordinationFactor - 1)) {
111    this.initialDemandList.add(new DemandBase(ActType.DESIGN, 3600, 9,
      17));
112   } else {
113    this.initialDemandList.add(new DemandBase(ActType.COORD, 3600, 9,
```

```
    17));
114   }
115  }
116  return initialDemandList;
117 }
118
119 private ListProperty<DemandFX> createInitialLeaderDemands(int
      numDemands) {
120  // Brute force method to synch tasks with supplies
121  for (int d = 0; d < .75 * numDemands; d++) {
122   this.initialDemandList.add(new DemandBase(ActType.WORK, 3600, 9, 17))
     ;
123  }
124  return initialDemandList;
125
126 }
127
128 // Method for creating more than 1 leader agent.
129 private void createLeaders(int numAgents) {
130  for (int e = 0; e < numAgents; e++) {
131   StringProperty _name = new SimpleStringProperty("Leader_" + (e + 1));
132   ObjectProperty<AgentType> _type = new SimpleObjectProperty<AgentType
     >(AgentType.LEADER);
133   agents.add(new Agent(_name, _type));
134  }
135 }
136
137 // Method for creating project staff engineers
138 private void createEngineers(int numAgents) {
139  /*
140   * 10 agents == defined test case 6 = high representation; 3 =
     moderate
141   * representation; 1 = low representation
142   */
143  int createdAgents = 0;
144
145  if (numAgents == 11) {
146   while (createdAgents < numAgents) {
147    if (createdAgents < 6) {
148    createdAgents += 1;
149     String _name = "Engineer_" + createdAgents;
150     agents.add(new Agent(_name, Race.HIGH_REP, Gender.HIGH_REP));
151    } else if (createdAgents < 9) {
152     createdAgents += 1;
153     String _name = "Engineer_" + createdAgents;
154     agents.add(new Agent(_name, Race.HIGH_REP, Gender.LOW_REP));
155    } else if (createdAgents < 10) {
156     createdAgents += 1;
157     String _name = "Engineer_" + createdAgents;
158     agents.add(new Agent(_name, Race.LOW_REP, Gender.LOW_REP));
159    } else {
160     createdAgents += 1;
161     agents.add(new Agent("Engineer_" + createdAgents, Race.LOW_REP,
     Gender.HIGH_REP));
```

```
    }
   }

  } else if (numAgents == 2) {
   agents.add(new Agent("Engineer_1", Race.HIGH_REP, Gender.LOW_REP));
   agents.add(new Agent("Engineer_2", Race.LOW_REP, Gender.HIGH_REP));
  } else {
   for (int e = 0; e < numAgents; e++) {
    StringProperty _name = new SimpleStringProperty("Engineer_" + (e +
    1));
    ObjectProperty<AgentType> _type = new SimpleObjectProperty<AgentType
    >(AgentType.ENGINEER);
    agents.add(new Agent(_name, _type));
   }
  }
 }

 public void resetSim() {
  agentEnvironment.lowRepAgentsRetained_timeSeries.clear();
  agentEnvironment.modRepAgentsRetained_timeSeries.clear();
  agentEnvironment.highRepAgentsRetained_timeSeries.clear();

  agentEnvironment.completedDemands_timeSeries.clear();
  agentEnvironment.diversityConcerns_timeSeries.clear();
  agentEnvironment.agentsAbsent_timeSeries.clear();
  agentEnvironment.coordRequests_timeSeries.clear();
  agentEnvironment.leadershipTasks_timeSeries.clear();
  agentEnvironment.intNetDensity_timeSeries.clear();
  agentEnvironment.intNetDensityLow_timeSeries.clear();
  agentEnvironment.intNetDensityMod_timeSeries.clear();
  agentEnvironment.intNetDensityHigh_timeSeries.clear();

  agentEnvironment.agents.clear();
  agentEnvironment.departedAgents.clear();
  agentEnvironment.globalAssignedDemands.clear();
  agentEnvironment.n_completeDiversityDemands.set(0);
  agentEnvironment.diversityConsequences.clear();

 }

 // Getters and setters
 public final DoubleProperty progressProperty() {
  return this.progress;
 }

 public final double getProgress() {
  return this.progressProperty().get();
 }

 public final void setProgress(final double progress) {
  this.progressProperty().set(progress);
 }

 public final BooleanProperty transformationalLeaderProperty() {
```

```
214   return this.transformationalLeader;
215  }
216
217  public final boolean isTransformationalLeader() {
218   return this.transformationalLeaderProperty().get();
219  }
220
221  public final void setTransformationalLeader(final boolean
       transformationalLeader) {
222   this.transformationalLeaderProperty().set(transformationalLeader);
223  }
224
225  public final BooleanProperty sociometricBadgeAwarenessProperty() {
226   return this.sociometricBadgeAwareness;
227  }
228
229  public final boolean isSociometricBadgeAwareness() {
230   return this.sociometricBadgeAwarenessProperty().get();
231  }
232
233  public final void setSociometricBadgeAwareness(final boolean
       sociometricBadgeAwareness) {
234   this.sociometricBadgeAwarenessProperty().set(sociometricBadgeAwareness
       );
235  }
236
237  public final ObjectProperty<DiversityApproach>
       diversityApproachProperty() {
238   return this.diversityApproach;
239  }
240
241  public final DiversityApproach getDiversityApproach() {
242   return this.diversityApproachProperty().get();
243  }
244
245  public final void setDiversityApproach(final DiversityApproach
       diversityApproach) {
246   this.diversityApproachProperty().set(diversityApproach);
247  }
248
249  public final IntegerProperty projectSpanProperty() {
250   return this.projectSpan;
251  }
252
253  public final int getProjectSpan() {
254   return this.projectSpanProperty().get();
255  }
256
257  public final void setProjectSpan(final int projectSpan) {
258   this.projectSpanProperty().set(projectSpan);
259  }
260
261  public final IntegerProperty agentTimeStepProperty() {
262   return this.agentTimeStep;
```

```
263 }
264
265 public final int getAgentTimeStep() {
266  return this.agentTimeStepProperty().get();
267 }
268
269 public final void setAgentTimeStep(final int agentTimeStep) {
270  this.agentTimeStepProperty().set(agentTimeStep);
271 }
272
273 public final IntegerProperty numLeadersProperty() {
274  return this.numLeaders;
275 }
276
277 public final int getNumLeaders() {
278  return this.numLeadersProperty().get();
279 }
280
281 public final void setNumLeaders(final int numLeaders) {
282  this.numLeadersProperty().set(numLeaders);
283 }
284
285 public final IntegerProperty numEngineersProperty() {
286  return this.numEngineers;
287 }
288
289 public final int getNumEngineers() {
290  return this.numEngineersProperty().get();
291 }
292
293 public final void setNumEngineers(final int numEngineers) {
294  this.numEngineersProperty().set(numEngineers);
295 }
296
297 public final ListProperty<Agent> agentsProperty() {
298  return this.agents;
299 }
300
301 public final ObservableList<Agent> getAgents() {
302  return this.agentsProperty().get();
303 }
304
305 public final void setAgents(final ObservableList<Agent> agents) {
306  this.agentsProperty().set(agents);
307 }
308
309 public final ListProperty<DemandFX> initialDemandListProperty() {
310  return this.initialDemandList;
311 }
312
313 public final ObservableList<DemandFX> getInitialDemandList() {
314  return this.initialDemandListProperty().get();
315 }
316
```

```
317  public final void setInitialDemandList(final ObservableList<DemandFX>
       initialDemandList) {
318   this.initialDemandListProperty().set(initialDemandList);
319  }
320
321 }
```

A.2.2 DiversityMonteCarloSim.java

```
1
2  import java.util.ArrayList;
3  import java.util.Comparator;
4
5  import javafx.beans.property.BooleanProperty;
6  import javafx.beans.property.DoubleProperty;
7  import javafx.beans.property.IntegerProperty;
8  import javafx.beans.property.ListProperty;
9  import javafx.beans.property.ObjectProperty;
10 import javafx.beans.property.SimpleBooleanProperty;
11 import javafx.beans.property.SimpleDoubleProperty;
12 import javafx.beans.property.SimpleIntegerProperty;
13 import javafx.beans.property.SimpleListProperty;
14 import javafx.beans.property.SimpleObjectProperty;
15 import javafx.collections.FXCollections;
16 import javafx.collections.ObservableList;
17 import javafx.scene.chart.XYChart.Data;
18
19 public class DiversityMonteCarloSim {
20
21  // States (potentially) controlled by User Interface
22  IntegerProperty numSims = new SimpleIntegerProperty(1);
23  IntegerProperty projectSpan;
24  IntegerProperty agentTimeStep;
25  IntegerProperty numLeaders;
26  IntegerProperty numEngineers;
27
28  ObjectProperty<DiversityApproach> diversityApproach;
29  BooleanProperty transformationalLeader;
30  BooleanProperty sociometricBadgeAwareness;
31
32  DoubleProperty progress;
33
34  ListProperty<Agent> agents;
35  ListProperty<SimResult> simResults;
36  ListProperty<DemandFX> initialDemandList;
37  World agentEnvironment;
38  DiversityModelRunner session;
39
40  // Values for/from UI
41  ListProperty<IntegerProperty> projectDuration = new SimpleListProperty
      <>(FXCollections.observableArrayList());
42  ListProperty<DoubleProperty> diversityDemands = new SimpleListProperty
      <>(FXCollections.observableArrayList());
43  ListProperty<DoubleProperty> diversityDemandsComplete = new
```

```
    SimpleListProperty<>(
44    FXCollections.observableArrayList());
45  ListProperty<DoubleProperty> intNetDensityStart = new
      SimpleListProperty<>(FXCollections.observableArrayList());
46  ListProperty<DoubleProperty> intNetDensityFinish = new
      SimpleListProperty<>(FXCollections.observableArrayList());
47
48  ArrayList<Double> indStart = new ArrayList<>();
49  ArrayList<Double> indDelta = new ArrayList<>();
50  // ListProperty<DoubleProperty> indDelta = new
51  // SimpleListProperty<>(FXCollections.observableArrayList());
52  ListProperty<DoubleProperty> numAgentsLostLow = new SimpleListProperty
      <>(FXCollections.observableArrayList());
53  ListProperty<DoubleProperty> numAgentsLostMod = new SimpleListProperty
      <>(FXCollections.observableArrayList());
54  ListProperty<DoubleProperty> numAgentsLostHigh = new SimpleListProperty
      <>(FXCollections.observableArrayList());
55 // ListProperty numLeaderInterventions = new SimpleListProperty<>(
      FXCollections.observableArrayList());
56 // ListProperty numLeaderMicroSupports = new SimpleListProperty<>(
      FXCollections.observableArrayList());
57
58  // Data Series for UI
59  public SimpleListProperty<Data<Number, Number>> indStart_series;
60  public SimpleListProperty<Data<Number, Number>> indFinal_series;
61  public SimpleListProperty<Data<Number, Number>>
      projectDuration_dataSeries;
62  public SimpleListProperty<Data<Number, Number>>
      numDiversityDemandsTotal_dataSeries;
63  public SimpleListProperty<Data<Number, Number>>
      numDiversityDemandsComplete_dataSeries;
64  public SimpleListProperty<Data<Number, Number>>
      numInteractions_dataSeries;
65  public SimpleListProperty<Data<Number, Number>>
      numInteractionsLow_dataSeries;
66  public SimpleListProperty<Data<Number, Number>>
      numInteractionsMod_dataSeries;
67  public SimpleListProperty<Data<Number, Number>>
      numInteractionsHigh_dataSeries;
68
69  public SimpleListProperty<Data<Number, Number>>
      numAgentAbsences_dataSeries;
70
71  public SimpleListProperty<Data<Number, Number>>
      numLeaderInterventions_dataSeries;
72  public SimpleListProperty<Data<Number, Number>>
      numLeaderMicroSupports_dataSeries;
73
74  public SimpleListProperty<Data<Number, Number>>
      numAgentsLostLow_dataSeries;
75  public SimpleListProperty<Data<Number, Number>>
      numAgentsLostMod_dataSeries;
76  public SimpleListProperty<Data<Number, Number>>
      numAgentsLostHigh_dataSeries;
```

```
77
78  public DiversityMonteCarloSim(int _numSims) {
79
80   // Create Initial Values for inclusion in UI
81   numSims = new SimpleIntegerProperty(1);
82   projectSpan = new SimpleIntegerProperty(90); // x-day project (TBR in
      UI)
83   numLeaders = new SimpleIntegerProperty(1); // x leader (TBR in UI)
84   numEngineers = new SimpleIntegerProperty(11); // x engineers (TBR in
      UI)
85   agentTimeStep = new SimpleIntegerProperty(1); // x day (TBR in UI)
86   progress = new SimpleDoubleProperty(0); // for progress bar
87
88   // Leadership parameters from UI for agent
89   transformationalLeader = new SimpleBooleanProperty(true);
90   sociometricBadgeAwareness = new SimpleBooleanProperty(true);
91   diversityApproach = new SimpleObjectProperty<DiversityApproach>(
      DiversityApproach.EQUALITY);
92
93   // Capture Results for UI
94   indStart_series = new SimpleListProperty<Data<Number, Number>>(
      FXCollections.observableArrayList());
95   indFinal_series = new SimpleListProperty<Data<Number, Number>>(
      FXCollections.observableArrayList());
96   projectDuration_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(FXCollections.observableArrayList());
97
98   numDiversityDemandsTotal_dataSeries = new SimpleListProperty<Data<
      Number, Number>>(
99     FXCollections.observableArrayList());
100  numDiversityDemandsComplete_dataSeries = new SimpleListProperty<Data<
      Number, Number>>(
101    FXCollections.observableArrayList());
102
103  numInteractions_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(FXCollections.observableArrayList());
104  numInteractionsLow_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(
105    FXCollections.observableArrayList());
106  numInteractionsMod_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(
107    FXCollections.observableArrayList());
108  numInteractionsHigh_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(
109    FXCollections.observableArrayList());
110
111  numAgentAbsences_dataSeries = new SimpleListProperty<Data<Number,
      Number>>(FXCollections.observableArrayList());
112
113  numLeaderInterventions_dataSeries = new SimpleListProperty<Data<Number
      , Number>>(
114    FXCollections.observableArrayList());
115  numLeaderMicroSupports_dataSeries = new SimpleListProperty<Data<Number
      , Number>>(
```

```
116     FXCollections.observableArrayList());
117
118   numAgentsLostLow_dataSeries = new SimpleListProperty<Data<Number,
        Number>>(FXCollections.observableArrayList());
119   numAgentsLostMod_dataSeries = new SimpleListProperty<Data<Number,
        Number>>(FXCollections.observableArrayList());
120   numAgentsLostHigh_dataSeries = new SimpleListProperty<Data<Number,
        Number>>(
121     FXCollections.observableArrayList());
122
123   // Set initial values for UI
124   setNumSims(10);
125
126
127 }
128
129 public void start() {
130   simResults = new SimpleListProperty<>(FXCollections.
        observableArrayList());
131
132   for (int i = 0; i < numSims.get(); i++) {
133
134     // Initiate Sim
135     DiversityModelRunner session = new DiversityModelRunner();
136     session.setProjectSpan(this.projectSpan.get());
137     session.setNumLeaders(this.numLeaders.get());
138     session.setNumEngineers(this.numEngineers.get());
139     session.setAgentTimeStep(this.agentTimeStep.get());
140     session.setTransformationalLeader(this.transformationalLeader.get());
141     session.setSociometricBadgeAwareness(this.sociometricBadgeAwareness.
        get());
142     session.setDiversityApproach(this.diversityApproach.get());
143     session.start();
144
145     // Capture Results
146     simResults.add(session.simResults);
147
148   }
149
150   int i = 0;
151   for (SimResult result : simResults) {
152
153     indDelta.add(result.getIntNetDensityFinish() - result.
        getIntNetDensityStart());
154     indStart_series.get().add(new Data<Number, Number>(i, result.
        getIntNetDensityStart()));
155     indFinal_series.get().add(new Data<Number, Number>(i, result.
        getIntNetDensityFinish()));
156     projectDuration_dataSeries.get().add(new Data<Number, Number>(i,
        result.projectDuration.get() / 28800));
157
158     numLeaderInterventions_dataSeries.get()
159       .add(new Data<Number, Number>(i, result.numLeaderInterventions.get
        ()));
```

```
160     numLeaderMicroSupports_dataSeries.get()
161       .add(new Data<Number, Number>(i, result.numLeaderMicroSupports.get
        ()));
162
163     numAgentAbsences_dataSeries.get().add(new Data<Number, Number>(i,
        result.agentAbsences.get()));
164     numDiversityDemandsTotal_dataSeries.get()
165       .add(new Data<Number, Number>(i, result.diversityDemandsGenerated.
        get()));
166     numDiversityDemandsComplete_dataSeries.get()
167       .add(new Data<Number, Number>(i, result.diversityDemandsComplete.
        get()));
168
169     numInteractions_dataSeries.get().add(new Data<Number, Number>(i,
        result.numInteractions.get()));
170
171     numAgentsLostLow_dataSeries.get().add(new Data<Number, Number>(i,
        result.numAgentsLostLow.get()));
172     numAgentsLostMod_dataSeries.get().add(new Data<Number, Number>(i,
        result.numAgentsLostMod.get()));
173     numAgentsLostHigh_dataSeries.get().add(new Data<Number, Number>(i,
        result.numAgentsLostHigh.get()));
174
175     i += 1;
176    }
177   }
178
179   // CONVENIENCE METHODS
180   class DoubleResultComparator implements Comparator<Double> {
181
182    @Override
183    public int compare(Double num1, Double num2) {
184     return Double.compare(num1, num2);
185    }
186
187   }
188
189   public void resetSim() {
190    indFinal_series.clear();
191    projectDuration_dataSeries.clear();
192    numDiversityDemandsTotal_dataSeries.clear();
193    numDiversityDemandsComplete_dataSeries.clear();
194    numAgentAbsences_dataSeries.clear();
195
196    numLeaderInterventions_dataSeries.clear();
197    numLeaderMicroSupports_dataSeries.clear();
198    numInteractions_dataSeries.clear();
199
200    numAgentsLostLow_dataSeries.clear();
201    numAgentsLostMod_dataSeries.clear();
202    numAgentsLostHigh_dataSeries.clear();
203   }
204
205   // GETTERS AND SETTERS
```

```
206 public final IntegerProperty projectSpanProperty() {
207  return this.projectSpan;
208 }
209
210 public final int getProjectSpan() {
211  return this.projectSpanProperty().get();
212 }
213
214 public final void setProjectSpan(final int projectSpan) {
215  this.projectSpanProperty().set(projectSpan);
216 }
217
218 public final IntegerProperty agentTimeStepProperty() {
219  return this.agentTimeStep;
220 }
221
222 public final int getAgentTimeStep() {
223  return this.agentTimeStepProperty().get();
224 }
225
226 public final void setAgentTimeStep(final int agentTimeStep) {
227  this.agentTimeStepProperty().set(agentTimeStep);
228 }
229
230 public final IntegerProperty numLeadersProperty() {
231  return this.numLeaders;
232 }
233
234 public final int getNumLeaders() {
235  return this.numLeadersProperty().get();
236 }
237
238 public final void setNumLeaders(final int numLeaders) {
239  this.numLeadersProperty().set(numLeaders);
240 }
241
242 public final IntegerProperty numEngineersProperty() {
243  return this.numEngineers;
244 }
245
246 public final int getNumEngineers() {
247  return this.numEngineersProperty().get();
248 }
249
250 public final void setNumEngineers(final int numEngineers) {
251  this.numEngineersProperty().set(numEngineers);
252 }
253
254 public final ObjectProperty<DiversityApproach>
      diversityApproachProperty() {
255  return this.diversityApproach;
256 }
257
258 public final DiversityApproach getDiversityApproach() {
```

```
259   return this.diversityApproachProperty().get();
260  }
261
262  public final void setDiversityApproach(final DiversityApproach
       diversityApproach) {
263   this.diversityApproachProperty().set(diversityApproach);
264  }
265
266  public final BooleanProperty transformationalLeaderProperty() {
267   return this.transformationalLeader;
268  }
269
270  public final boolean isTransformationalLeader() {
271   return this.transformationalLeaderProperty().get();
272  }
273
274  public final void setTransformationalLeader(final boolean
       transformationalLeader) {
275   this.transformationalLeaderProperty().set(transformationalLeader);
276  }
277
278  public final BooleanProperty sociometricBadgeAwarenessProperty() {
279   return this.sociometricBadgeAwareness;
280  }
281
282  public final boolean isSociometricBadgeAwareness() {
283   return this.sociometricBadgeAwarenessProperty().get();
284  }
285
286  public final void setSociometricBadgeAwareness(final boolean
       sociometricBadgeAwareness) {
287   this.sociometricBadgeAwarenessProperty().set(sociometricBadgeAwareness
       );
288  }
289
290  public final DoubleProperty progressProperty() {
291   return this.progress;
292  }
293
294  public final double getProgress() {
295   return this.progressProperty().get();
296  }
297
298  public final void setProgress(final double progress) {
299   this.progressProperty().set(progress);
300  }
301
302  public final ListProperty<Agent> agentsProperty() {
303   return this.agents;
304  }
305
306  public final ObservableList<Agent> getAgents() {
307   return this.agentsProperty().get();
308  }
```

```
309
310 public final void setAgents(final ObservableList<Agent> agents) {
311  this.agentsProperty().set(agents);
312 }
313
314 public final ListProperty<DemandFX> initialDemandListProperty() {
315  return this.initialDemandList;
316 }
317
318 public final ObservableList<DemandFX> getInitialDemandList() {
319  return this.initialDemandListProperty().get();
320 }
321
322 public final void setInitialDemandList(final ObservableList<DemandFX>
      initialDemandList) {
323  this.initialDemandListProperty().set(initialDemandList);
324 }
325
326 public final IntegerProperty numSimsProperty() {
327  return this.numSims;
328 }
329
330 public final int getNumSims() {
331  return this.numSimsProperty().get();
332 }
333
334 public final void setNumSims(final int numSims) {
335  this.numSimsProperty().set(numSims);
336 }
337
338 public final SimpleListProperty<Data<Number, Number>>
      ind_delta_seriesProperty() {
339  return this.indFinal_series;
340 }
341
342 public final ObservableList<Data<Number, Number>> getInd_delta_series()
       {
343  return this.ind_delta_seriesProperty().get();
344 }
345
346 public final void setInd_delta_series(final ObservableList<Data<Number,
       Number>> ind_delta_series) {
347  this.ind_delta_seriesProperty().set(ind_delta_series);
348 }
349
350 public final SimpleListProperty<Data<Number, Number>>
      projectDuration_dataSeriesProperty() {
351  return this.projectDuration_dataSeries;
352 }
353
354 public final ObservableList<Data<Number, Number>>
      getProjectDuration_dataSeries() {
355  return this.projectDuration_dataSeriesProperty().get();
356 }
```

```
357
358 public final void setProjectDuration_dataSeries(
359   final ObservableList<Data<Number, Number>> projectDuration_dataSeries
      ) {
360  this.projectDuration_dataSeriesProperty().set(
      projectDuration_dataSeries);
361 }
362
363 public final SimpleListProperty<Data<Number, Number>>
      numDiversityDemandsComplete_dataSeriesProperty() {
364  return this.numDiversityDemandsComplete_dataSeries;
365 }
366
367 public final ObservableList<Data<Number, Number>>
      getNumDiversityDemandsComplete_dataSeries() {
368  return this.numDiversityDemandsComplete_dataSeriesProperty().get();
369 }
370
371 public final void setNumDiversityDemandsComplete_dataSeries(
372   final ObservableList<Data<Number, Number>>
      numDiversityDemandsComplete_dataSeries) {
373  this.numDiversityDemandsComplete_dataSeriesProperty().set(
      numDiversityDemandsComplete_dataSeries);
374 }
375
376 public final SimpleListProperty<Data<Number, Number>>
      numAgentsLostLow_dataSeriesProperty() {
377  return this.numAgentsLostLow_dataSeries;
378 }
379
380 public final ObservableList<Data<Number, Number>>
      getNumAgentsLostLow_dataSeries() {
381  return this.numAgentsLostLow_dataSeriesProperty().get();
382 }
383
384 public final void setNumAgentsLostLow_dataSeries(
385   final ObservableList<Data<Number, Number>>
      numAgentsLostLow_dataSeries) {
386  this.numAgentsLostLow_dataSeriesProperty().set(
      numAgentsLostLow_dataSeries);
387 }
388
389 public final SimpleListProperty<Data<Number, Number>>
      numAgentsLostMod_dataSeriesProperty() {
390  return this.numAgentsLostMod_dataSeries;
391 }
392
393 public final ObservableList<Data<Number, Number>>
      getNumAgentsLostMod_dataSeries() {
394  return this.numAgentsLostMod_dataSeriesProperty().get();
395 }
396
397 public final void setNumAgentsLostMod_dataSeries(
398   final ObservableList<Data<Number, Number>>
```

```
    numAgentsLostMod_dataSeries) {
399  this.numAgentsLostMod_dataSeriesProperty().set(
    numAgentsLostMod_dataSeries);
400 }
401
402 public final SimpleListProperty<Data<Number, Number>>
    numAgentsLostHigh_dataSeriesProperty() {
403  return this.numAgentsLostHigh_dataSeries;
404 }
405
406 public final ObservableList<Data<Number, Number>>
    getNumAgentsLostHigh_dataSeries() {
407  return this.numAgentsLostHigh_dataSeriesProperty().get();
408 }
409
410 public final void setNumAgentsLostHigh_dataSeries(
411   final ObservableList<Data<Number, Number>>
    numAgentsLostHigh_dataSeries) {
412  this.numAgentsLostHigh_dataSeriesProperty().set(
    numAgentsLostHigh_dataSeries);
413 }
414
415 public final ListProperty<SimResult> simResultsProperty() {
416  return this.simResults;
417 }
418
419 public final ObservableList<SimResult> getSimResults() {
420  return this.simResultsProperty().get();
421 }
422
423 public final void setSimResults(final ObservableList<SimResult>
    simResults) {
424  this.simResultsProperty().set(simResults);
425 }
426
427 public final ListProperty<IntegerProperty> projectDurationProperty() {
428  return this.projectDuration;
429 }
430
431 public final ObservableList<IntegerProperty> getProjectDuration() {
432  return this.projectDurationProperty().get();
433 }
434
435 public final void setProjectDuration(final ObservableList<
    IntegerProperty> projectDuration) {
436  this.projectDurationProperty().set(projectDuration);
437 }
438
439 public final ListProperty<DoubleProperty>
    diversityDemandsCompleteProperty() {
440  return this.diversityDemandsComplete;
441 }
442
443 public final ObservableList<DoubleProperty> getDiversityDemandsComplete
```

```
    () {
444  return this.diversityDemandsCompleteProperty().get();
445 }
446
447 public final void setDiversityDemandsComplete(final ObservableList<
      DoubleProperty> diversityDemandsComplete) {
448  this.diversityDemandsCompleteProperty().set(diversityDemandsComplete);
449 }
450
451 public final ListProperty<DoubleProperty> intNetDensityStartProperty()
      {
452  return this.intNetDensityStart;
453 }
454
455 public final ObservableList<DoubleProperty> getIntNetDensityStart() {
456  return this.intNetDensityStartProperty().get();
457 }
458
459 public final void setIntNetDensityStart(final ObservableList<
      DoubleProperty> intNetDensityStart) {
460  this.intNetDensityStartProperty().set(intNetDensityStart);
461 }
462
463 public final ListProperty<DoubleProperty> intNetDensityFinishProperty()
      {
464  return this.intNetDensityFinish;
465 }
466
467 public final ObservableList<DoubleProperty> getIntNetDensityFinish() {
468  return this.intNetDensityFinishProperty().get();
469 }
470
471 public final void setIntNetDensityFinish(final ObservableList<
      DoubleProperty> intNetDensityFinish) {
472  this.intNetDensityFinishProperty().set(intNetDensityFinish);
473 }
474
475 public final ListProperty<DoubleProperty> numAgentsLostLowProperty() {
476  return this.numAgentsLostLow;
477 }
478
479 public final ObservableList<DoubleProperty> getNumAgentsLostLow() {
480  return this.numAgentsLostLowProperty().get();
481 }
482
483 public final void setNumAgentsLostLow(final ObservableList<
      DoubleProperty> numAgentsLostLow) {
484  this.numAgentsLostLowProperty().set(numAgentsLostLow);
485 }
486
487 public final ListProperty<DoubleProperty> numAgentsLostModProperty() {
488  return this.numAgentsLostMod;
489 }
490
```

```
491 public final ObservableList<DoubleProperty> getNumAgentsLostMod() {
492  return this.numAgentsLostModProperty().get();
493 }
494
495 public final void setNumAgentsLostMod(final ObservableList<
      DoubleProperty> numAgentsLostMod) {
496  this.numAgentsLostModProperty().set(numAgentsLostMod);
497 }
498
499 public final ListProperty<DoubleProperty> numAgentsLostHighProperty() {
500  return this.numAgentsLostHigh;
501 }
502
503 public final ObservableList<DoubleProperty> getNumAgentsLostHigh() {
504  return this.numAgentsLostHighProperty().get();
505 }
506
507 public final void setNumAgentsLostHigh(final ObservableList<
      DoubleProperty> numAgentsLostHigh) {
508  this.numAgentsLostHighProperty().set(numAgentsLostHigh);
509 }
510
511 public final ListProperty<DoubleProperty> diversityDemandsProperty() {
512  return this.diversityDemands;
513 }
514
515 public final ObservableList<DoubleProperty> getDiversityDemands() {
516  return this.diversityDemandsProperty().get();
517 }
518
519 public final void setDiversityDemands(final ObservableList<
      DoubleProperty> diversityDemands) {
520  this.diversityDemandsProperty().set(diversityDemands);
521 }
522
523 public final SimpleListProperty<Data<Number, Number>>
      numDiversityDemandsTotal_dataSeriesProperty() {
524  return this.numDiversityDemandsTotal_dataSeries;
525 }
526
527 public final ObservableList<Data<Number, Number>>
      getNumDiversityDemandsTotal_dataSeries() {
528  return this.numDiversityDemandsTotal_dataSeriesProperty().get();
529 }
530
531 public final void setNumDiversityDemandsTotal_dataSeries(
532   final ObservableList<Data<Number, Number>>
      numDiversityDemandsTotal_dataSeries) {
533  this.numDiversityDemandsTotal_dataSeriesProperty().set(
      numDiversityDemandsTotal_dataSeries);
534 }
535
536 public final SimpleListProperty<Data<Number, Number>>
      numAgentAbsences_dataSeriesProperty() {
```

```
537   return this.numAgentAbsences_dataSeries;
538  }
539
540  public final ObservableList<Data<Number, Number>>
       getNumAgentAbsences_dataSeries() {
541   return this.numAgentAbsences_dataSeriesProperty().get();
542  }
543
544  public final void setNumAgentAbsences_dataSeries(
545    final ObservableList<Data<Number, Number>>
       numAgentAbsences_dataSeries) {
546   this.numAgentAbsences_dataSeriesProperty().set(
       numAgentAbsences_dataSeries);
547  }
548
549  public final SimpleListProperty<Data<Number, Number>>
       numLeaderInterventions_dataSeriesProperty() {
550   return this.numLeaderInterventions_dataSeries;
551  }
552
553  public final ObservableList<Data<Number, Number>>
       getNumLeaderInterventions_dataSeries() {
554   return this.numLeaderInterventions_dataSeriesProperty().get();
555  }
556
557  public final void setNumLeaderInterventions_dataSeries(
558    final ObservableList<Data<Number, Number>>
       numLeaderInterventions_dataSeries) {
559   this.numLeaderInterventions_dataSeriesProperty().set(
       numLeaderInterventions_dataSeries);
560  }
561
562  public final SimpleListProperty<Data<Number, Number>>
       numLeaderMicroSupports_dataSeriesProperty() {
563   return this.numLeaderMicroSupports_dataSeries;
564  }
565
566  public final ObservableList<Data<Number, Number>>
       getNumLeaderMicroSupports_dataSeries() {
567   return this.numLeaderMicroSupports_dataSeriesProperty().get();
568  }
569
570  public final void setNumLeaderMicroSupports_dataSeries(
571    final ObservableList<Data<Number, Number>>
       numLeaderMicroSupports_dataSeries) {
572   this.numLeaderMicroSupports_dataSeriesProperty().set(
       numLeaderMicroSupports_dataSeries);
573  }
574
575  public final SimpleListProperty<Data<Number, Number>>
       numInteractions_dataSeriesProperty() {
576   return this.numInteractions_dataSeries;
577  }
578
```

```
579  public final ObservableList<Data<Number, Number>>
       getNumInteractions_dataSeries() {
580   return this.numInteractions_dataSeriesProperty().get();
581  }
582
583  public final void setNumInteractions_dataSeries(
584    final ObservableList<Data<Number, Number>> numInteractions_dataSeries
       ) {
585   this.numInteractions_dataSeriesProperty().set(
       numInteractions_dataSeries);
586  }
587
588  public final SimpleListProperty<Data<Number, Number>>
       indFinal_seriesProperty() {
589   return this.indFinal_series;
590  }
591
592  public final ObservableList<Data<Number, Number>> getIndFinal_series()
       {
593   return this.indFinal_seriesProperty().get();
594  }
595
596  public final void setIndFinal_series(final ObservableList<Data<Number,
       Number>> indFinal_series) {
597   this.indFinal_seriesProperty().set(indFinal_series);
598  }
599
600  public final SimpleListProperty<Data<Number, Number>>
       indStart_seriesProperty() {
601   return this.indStart_series;
602  }
603
604  public final ObservableList<Data<Number, Number>> getIndStart_series()
       {
605   return this.indStart_seriesProperty().get();
606  }
607
608  public final void setIndStart_series(final ObservableList<Data<Number,
       Number>> indStart_series) {
609   this.indStart_seriesProperty().set(indStart_series);
610  }
611
612 }
```

A.2.3 World.java

```
1
2 import java.util.Comparator;
3 import java.util.Random;
4
5 import javafx.beans.property.DoubleProperty;
6 import javafx.beans.property.IntegerProperty;
7 import javafx.beans.property.ListProperty;
8 import javafx.beans.property.ObjectProperty;
```

```
9  import javafx.beans.property.SimpleDoubleProperty;
10 import javafx.beans.property.SimpleIntegerProperty;
11 import javafx.beans.property.SimpleListProperty;
12 import javafx.beans.property.SimpleObjectProperty;
13 import javafx.beans.property.SimpleStringProperty;
14 import javafx.beans.property.StringProperty;
15 import javafx.collections.FXCollections;
16 import javafx.collections.ObservableList;
17 import javafx.scene.chart.XYChart.Data;
18
19 /**
20  * \
21  *
22  * @author Jeremy Goodwin with considerable support from Dr. Bryan Moser
       Written
23  *         in conjunction with thesis pursuant to MIT SDM degree, 2021
24  *
25  *         This class serves as the environment in which agents operate.
26  *
27  */
28 public class World {
29
30  // ENVIRONMENT PARAMETERS
31  StringProperty name; // Environment name
32  IntegerProperty globalTime = new SimpleIntegerProperty(0); //
      Environment time stamp
33  IntegerProperty globalTimeDays = new SimpleIntegerProperty(0); // for
      UI
34
35  // AGENT LISTS
36  ListProperty<Agent> agents = new SimpleListProperty<>(FXCollections.
      observableArrayList());
37  ListProperty<Agent> departedAgents = new SimpleListProperty<>(
      FXCollections.observableArrayList());
38
39  // AGENT PARAMETERS FOR/FROM UI
40  IntegerProperty agentTimeStep;
41  DoubleProperty teamIntNetDensity = new SimpleDoubleProperty(0.0); //
      internal net density
42  DoubleProperty teamIntNetDensityLow = new SimpleDoubleProperty(0.0); //
       internal net density
43  DoubleProperty teamIntNetDensityMod = new SimpleDoubleProperty(0.0); //
       internal net density
44  DoubleProperty teamIntNetDensityHigh = new SimpleDoubleProperty(0.0);
      // internal net density
45
46  // PROJECT PARAMETERS FOR/FROM UI
47  IntegerProperty projectSpan;
48  ListProperty<DemandFX> globalAssignedDemands = new SimpleListProperty<
      DemandFX>();
49  ObjectProperty<DiversityApproach> diversityApproach = new
      SimpleObjectProperty<DiversityApproach>();
50
51  // Emerging project parameters
```

```
52 ListProperty<DemandFX> diversityConsequences;
53 IntegerProperty n_completeDemands = new SimpleIntegerProperty(0);
54 DoubleProperty n_diversityDemands = new SimpleDoubleProperty(0);
55 DoubleProperty n_completeDiversityDemands = new SimpleDoubleProperty(0)
     ;
56 IntegerProperty n_agentAbsences = new SimpleIntegerProperty(0);
57 DoubleProperty n_agentsLost_low = new SimpleDoubleProperty(0);
58 DoubleProperty n_agentsLost_moderate = new SimpleDoubleProperty(0);
59 DoubleProperty n_agentsLost_high = new SimpleDoubleProperty(0);
60 IntegerProperty n_leaderInterventions = new SimpleIntegerProperty(0);
61 IntegerProperty n_leaderMicroSupports = new SimpleIntegerProperty(0);
62 IntegerProperty n_interactions = new SimpleIntegerProperty(0);
63 IntegerProperty n_interactionsLow = new SimpleIntegerProperty(0);
64 IntegerProperty n_interactionsMod = new SimpleIntegerProperty(0);
65 IntegerProperty n_interactionsHigh = new SimpleIntegerProperty(0);
66 DoubleProperty indStart = new SimpleDoubleProperty(0);
67 DoubleProperty indFinish = new SimpleDoubleProperty(0);
68
69 ListProperty<DemandFX> globalCoordRequests = new SimpleListProperty<
     DemandFX>();
70 ListProperty<DemandFX> globalEmergingSignals = new SimpleListProperty<
     DemandFX>();
71
72 // Lists of important results for user interface
73 public SimpleListProperty<Data<Number, Number>>
     lowRepAgentsRetained_timeSeries;
74 public SimpleListProperty<Data<Number, Number>>
     modRepAgentsRetained_timeSeries;
75 public SimpleListProperty<Data<Number, Number>>
     highRepAgentsRetained_timeSeries;
76 public SimpleListProperty<Data<Number, Number>>
     completedDemands_timeSeries;
77 public SimpleListProperty<Data<Number, Number>>
     diversityConcerns_timeSeries;
78 public SimpleListProperty<Data<Number, Number>> agentsAbsent_timeSeries
     ;
79 public SimpleListProperty<Data<Number, Number>>
     leaderMicroSupports_timeSeries;
80 public SimpleListProperty<Data<Number, Number>>
     intNetDensity_timeSeries;
81 public SimpleListProperty<Data<Number, Number>>
     intNetDensityLow_timeSeries;
82 public SimpleListProperty<Data<Number, Number>>
     intNetDensityMod_timeSeries;
83 public SimpleListProperty<Data<Number, Number>>
     intNetDensityHigh_timeSeries;
84
85 public SimpleListProperty<Data<Number, Number>>
     coordRequests_timeSeries;
86 public SimpleListProperty<Data<Number, Number>>
     leadershipTasks_timeSeries;
87
88 // CONSTRUCTOR
89 public World() {
```

```
90      this(new SimpleStringProperty("World"), new SimpleListProperty<>(
          FXCollections.observableArrayList()));
91    }
92
93    // FULL CONSTRUCTOR
94    public World(SimpleStringProperty _name, SimpleListProperty<Agent>
          _agents) {
95     name = _name;
96     agents = _agents;
97
98     // Initialize lists for tracking results in UI
99     diversityConsequences = new SimpleListProperty<DemandFX>(FXCollections
          .observableArrayList());
100
101    lowRepAgentsRetained_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
102      FXCollections.observableArrayList());
103    modRepAgentsRetained_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
104      FXCollections.observableArrayList());
105    highRepAgentsRetained_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
106      FXCollections.observableArrayList());
107
108    completedDemands_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(FXCollections.observableArrayList());
109    leadershipTasks_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(FXCollections.observableArrayList());
110
111    diversityConcerns_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
112      FXCollections.observableArrayList());
113    agentsAbsent_timeSeries = new SimpleListProperty<Data<Number, Number
          >>(FXCollections.observableArrayList());
114
115    leaderMicroSupports_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
116      FXCollections.observableArrayList());
117    intNetDensity_timeSeries = new SimpleListProperty<Data<Number, Number
          >>(FXCollections.observableArrayList());
118    intNetDensityLow_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(FXCollections.observableArrayList());
119    intNetDensityMod_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(FXCollections.observableArrayList());
120    intNetDensityHigh_timeSeries = new SimpleListProperty<Data<Number,
          Number>>(
121      FXCollections.observableArrayList());
122
123    coordRequests_timeSeries = new SimpleListProperty<Data<Number, Number
          >>(FXCollections.observableArrayList());
124   }
125
126   // EXECUTION CONTROL METHODS
127   public void start() {
```

```
128
129   // Print initialization results to screen
130   printStartLoop(false);
131
132   // Calculate initial team IND
133   teamIntNetDensity = refreshTeamIntlNetworkDensity(true);
134
135   // Show initial stats in UI
136   updateUI(false);
137
138   // Pass initial demand list to agent & start simulation
139   agents.stream().forEach(a -> {
140    a.setInternalAssignedDemands(this.globalAssignedDemands);
141    a.start();
142    globalAssignedDemands.addAll(a.internalEmergingSignals);
143   });
144
145   // Summarize Day 1
146   setGlobalTime(agents.stream().min(new AgentTimeComparator()).get().
       agentTime.get());
147
148   // printDemandStatus();
149   updateUI(false);
150
151   // TODO: switch this on to allow Sim to continue running
152   continueSim();
153  }
154
155  public void continueSim() {
156
157   // print current status to screen
158   printStartLoop(false);
159
160   // continue if demands remain
161
162   if (this.checkAnyDemandList(globalAssignedDemands) && agents.size() >
       1) {
163
164    teamIntNetDensity = refreshTeamIntlNetworkDensity(false);
165    for (Agent agent : agents) {
166
167     // back to work
168     agent.continueSim();
169
170     // collect diversity consequences for UI
171     if (!agent.diversityConsequences.isEmpty()) {
172      for (DemandFX demand : agent.diversityConsequences) {
173       if (!this.diversityConsequences.contains(demand)) {
174        this.diversityConsequences.add(demand);
175       }
176      }
177     }
178
179    }
```

```
180   }
181
182   // collect departed agents and remove them from operation
183   retireAgents();
184
185   // refresh global time
186   setGlobalTime(agents.stream().min(new AgentTimeComparator()).get().
       agentTime.get());
187
188   // update UI
189   updateUI(false);
190   // printDemandStatus();
191
192   // return absent (not departed) agents for next period
193   refreshAgents();
194
195   if (this.checkAnyDemandList(globalAssignedDemands) && !agents.isEmpty
       ()) {
196    continueSim();
197   } else {
198 //    System.out.println("---------- SIMULATION COMPLETE ---------- ");
199 //    printSimResults(true, agents);
200 //    printSimResults(true, departedAgents);
201
202   }
203  }
204
205  // OUTPUT CONTROL METHODS
206  private void printStartLoop(boolean firstPrint) {
207
208   if (firstPrint) {
209    System.out.println("---------- Initialization Report Start ----------
       ");
210    System.out.println(name.get() + " Initiated with " +
       diversityApproach.get() + " diversity approach! ");
211    System.out.println("World time: " + globalTime.get());
212
213    System.out.println("Number of initial tasks: " +
       globalAssignedDemands.size());
214    System.out.println("Number of coord-required tasks: "
215      + globalAssignedDemands.stream().filter(d -> d.getType() == ActType
       .COORD).count());
216    System.out.println("Number of leader tasks: "
217      + globalAssignedDemands.stream().filter(d -> d.getType() == ActType
       .WORK).count());
218    System.out.println("Number of initial agents: " + agents.size());
219
220    System.out.println("\n*** Agent demographics ***");
221    agents.stream().forEach(a -> System.out.println(a.name.get() + " " +
       a.race.get() + " " + a.gender.get()));
222
223    System.out.println("\n*** Demographic Summary ***");
224    System.out.println("High Representation    : " + countAgents("high"))
       ;
```

```
225     System.out.println("Mod Representation     : " + countAgents("
        moderate"));
226     System.out.println("Low Representation     : " + countAgents("low"));
227     System.out.println("---------- Initialization Report Complete
        ----------\n");
228   } else {
229 //    System.out.println("\nDay time: " + globalTime.get() / 28800);
230 //    System.out.println("Number of agents departed this loop: "
231 //      + agents.stream().filter(agent -> agent.mode.get() == AgentMode.
      DEPARTED).count());
232   }
233  }
234
235  private void printDemandStatus() {
236   System.out.println("\nGlobal Demands Status:");
237
238   System.out.println("Inactive: " + countStateDemands(DemandState.
       INACTIVE));
239   System.out.println("Pending: " + countStateDemands(DemandState.PENDING
       ));
240   System.out.println("Started: " + countStateDemands(DemandState.STARTED
       ));
241   System.out.println("Active: " + countStateDemands(DemandState.ACTIVE))
       ;
242   System.out.println("Complete: " + countStateDemands(DemandState.
       COMPLETE) + " including "
243     + countDiversityConsequences() + " diversity consequences.");
244   System.out.println("Ignored: " + countStateDemands(DemandState.IGNORED
       ));
245   System.out.println("Ignored: " + countStateDemands(DemandState.IGNORED
       ));
246
247   System.out.println("\nDesign Remaining: " + countTypeDemandsRemaining(
       ActType.DESIGN));
248   System.out.println("Work Remaining: " + countTypeDemandsRemaining(
       ActType.WORK));
249   System.out.println("Lead Remaining: " + countTypeDemandsRemaining(
       ActType.LEAD));
250
251  }
252
253  public void updateUI(boolean printResults) {
254
255   // Update UI charts
256   lowRepAgentsRetained_timeSeries.get()
257     .add(new Data<Number, Number>(globalTime.get() / 28800, countAgents(
       "low")));
258   modRepAgentsRetained_timeSeries.get()
259     .add(new Data<Number, Number>(globalTime.get() / 28800, countAgents(
       "moderate")));
260   highRepAgentsRetained_timeSeries.get()
261     .add(new Data<Number, Number>(globalTime.get() / 28800, countAgents(
       "high")));
262
```

```
263   diversityConcerns_timeSeries.get()
264     .add(new Data<Number, Number>(globalTime.get() / 28800,
      countDiversityConsequences()));
265
266   agentsAbsent_timeSeries.get().add(new Data<Number, Number>(globalTime.
      get() / 28800, countAllAbsences()));
267
268   completedDemands_timeSeries.get().add(new Data<Number, Number>(
      globalTime.get() / 28800,
269     globalAssignedDemands.size() - countStateDemands(DemandState.
      COMPLETE)));
270
271   coordRequests_timeSeries.get()
272     .add(new Data<Number, Number>(globalTime.get() / 28800,
      countCoordRequests("all")));
273
274   leadershipTasks_timeSeries.get()
275     .add(new Data<Number, Number>(globalTime.get() / 28800,
      countLeadershipTasks()));
276
277   intNetDensity_timeSeries.get().add(new Data<Number, Number>(globalTime
      .get() / 28800, teamIntNetDensity.get()));
278   intNetDensityLow_timeSeries.get()
279     .add(new Data<Number, Number>(globalTime.get() / 28800,
      teamIntNetDensityLow.get()));
280   intNetDensityMod_timeSeries.get()
281     .add(new Data<Number, Number>(globalTime.get() / 28800,
      teamIntNetDensityMod.get()));
282   intNetDensityHigh_timeSeries.get()
283     .add(new Data<Number, Number>(globalTime.get() / 28800,
      teamIntNetDensityHigh.get()));
284
285   // Update Text Fields
286   setGlobalTimeDays(getGlobalTime());
287   setN_completeDemands(globalAssignedDemands.size());
288   setN_diversityDemands(diversityConsequences.size());
289   setN_completeDiversityDemands(
290     diversityConsequences.stream().filter(d -> d.getState() ==
      DemandState.COMPLETE).count());
291
292   // setTeamIntNetDensity(getTeamIntNetDensity());
293   // setTeamIntNetDensityLow(getTeamIntNetDensityLow());
294   // setTeamIntNetDensityMod(getTeamIntNetDensityMod());
295   // setTeamIntNetDensityHigh(getTeamIntNetDensityHigh());
296   double startingIND = new SimpleDoubleProperty(
297     (double) this.intNetDensity_timeSeriesProperty().get(0).getYValue())
      .get();
298
299   double finishIND = new SimpleDoubleProperty((double)
      intNetDensity_timeSeriesProperty()
300     .get(intNetDensity_timeSeriesProperty().size() - 1).getYValue()).get
      ();
301   setIndStart(startingIND);
302   setIndFinish(finishIND);
```

```
303
304   setN_agentsLost_low(departedAgents.stream().filter(a -> a.
       lowRepresentation.get() == true).count());
305   setN_agentsLost_moderate(departedAgents.stream().filter(a -> a.
       moderateRepresentation.get() == true).count());
306   setN_agentsLost_high(departedAgents.stream()
307     .filter(a -> a.lowRepresentation.get() != true && a.
       moderateRepresentation.get() != true).count());
308
309   // Other Monte Carlo Results Tracking
310   setN_agentAbsences(countAllAbsences());
311   setN_leaderInterventions(countLeaderInterventions());
312   setN_leaderMicroSupports(countLeaderMicroSupports());
313   setN_interactions(countCoordRequests("all"));
314
315   // Update screen output
316   int testTime = agents.stream().min(new AgentTimeComparator()).get().
       agentTime.get();
317
318   if (printResults) {
319    System.out.println("\n*** Results: ***");
320    System.out.println("Min agent time: " + testTime / 28800);
321
322    for (Agent agent : agents) {
323     System.out.println("\n" + agent.name.get() + " Current Agent Time: "
       + agent.agentTime.get() / 28000);
324     System.out.println("Current mode: " + agent.mode.get());
325     System.out.println("I completed " + agent.internalDemandsMoot.size()
       + " demands.");
326     System.out.println("I completed "
327       + agent.internalDemandsMoot.stream().filter(d -> d.getType() ==
       ActType.DESIGN)
328         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
       count()
329       + " Design Demands");
330     System.out.println("I completed "
331       + agent.internalDemandsMoot.stream().filter(d -> d.getType() ==
       ActType.LEAD)
332         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
       count()
333       + " Leadership Demands");
334     System.out.println("I completed "
335       + agent.internalDemandsMoot.stream().filter(d -> d.getType() ==
       ActType.COORD)
336         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
       count()
337       + " Coordination Demands");
338     System.out.println("My calls for help: " + agent.
       internalEmergingSignals.size());
339     System.out.println("My felt consequences due to diversity: " + agent
       .diversityConsequences.size());
340     System.out.println("My number of days off: " + agent.
       getAgentAbsences());
341     System.out.println("Acting Time: " + agent.agentTimeActing.get());
```

```
342     }
343
344     System.out.println("New Global Time: " + globalTime.get());
345    }
346
347   }
348
349   private void printSimResults(boolean print, ListProperty<Agent> _agents
        ) {
350    if (print) {
351     System.out.println("Project Length:                " + this.
        getGlobalTime() / 28800);
352     System.out.println("Team Starting Int Net Density: " + this.indStart.
        get());
353     System.out.println("Team Finishing Net Density:    " + this.
        getTeamIntNetDensity());
354     System.out.println();
355     for (Agent agent : _agents) {
356      System.out.println("Agent:               " + agent.getName());
357      System.out.println("Low Representation: " + agent.lowRepresentation.
        get());
358      System.out.println("Mod Representation: " + agent.
        moderateRepresentation.get());
359      System.out.println("Demands Complete:   " + agent.
        internalDemandsMoot.size());
360      System.out.println("Diversity Concerns: " + agent.
        diversityConsequences.size());
361      System.out.println("Days Off:           " + agent.getAgentAbsences()
        );
362      System.out.println();
363     }
364    }
365   }
366
367   // AGENT CONVENIENCE METHODS
368   private void refreshAgents() {
369    agents.stream().filter(a -> a.mode.get() != AgentMode.DEPARTED).
        forEach(a -> a.mode.set(AgentMode.INACTIVE));
370   }
371
372   private void retireAgents() {
373    for (Agent agent : agents) {
374     if (agent.mode.get() == AgentMode.DEPARTED) {
375      if (!departedAgents.contains(agent)) {
376       departedAgents.add(agent);
377      }
378     }
379    }
380
381    if (!departedAgents.isEmpty()) {
382     for (Agent agent : departedAgents) {
383      agents.remove(agent);
384     }
385    }
```

```
386 }
387
388 private double getAgentConnection(Agent agent1, Agent agent2) {
389  double connection;
390
391  // Assumption: range from 0.25-0.95
392  double minValue = 0.25;
393  double maxValue = 0.95;
394
395  // Assumption: bias for similar race and/or gender to account for
      similarity /
396  // homophily;
397  double bias1 = 0.25;
398  double bias2 = 0.125;
399
400  Random rand = new Random();
401
402  if (agent1 == agent2) {
403   connection = 1;
404
405  } else if (agent1.race.get() == agent2.race.get() && agent1.gender.get
      () == agent2.gender.get()) {
406
407   // assumption: higher connection due to similar demographics
408   connection = Math.min(bias1 + minValue + rand.nextDouble(), maxValue)
      ;
409
410  } else if (agent1.race.get() == agent2.race.get() || agent1.gender.get
      () == agent2.gender.get()) {
411
412   // assumption: higher connection due to similar demographics
413   connection = Math.min(bias2 + minValue + rand.nextDouble(), maxValue)
      ;
414
415  } else {
416   connection = Math.min(minValue + rand.nextDouble(), maxValue);
417  }
418  return connection;
419 }
420
421 private DoubleProperty refreshTeamIntlNetworkDensity(boolean initial) {
422  /*
423   * z_ij = internal network densities from agent_1 to agent_2 z_iq =
      max tie of
424   * agent 1 to anyone on team
425   *
426   * calculation from Reagans et al, 2004: ind = sum(sum((z_ij / z_iq)/(
      n(n-1))))
427   * initialRun allows us to build initial vector, and then update based
       on model
428   * dynamics -- note: model intentionally not scaled by z_iq as that
      operation
429   * was done by Reagans et al to account for inaccuracies in human
      reporting
```

```
430     *
431     */
432
433    // internal variables
434    double z_ij;
435    double z_iq;
436    long n = agents.stream().filter(a -> a.mode.get() != AgentMode.
        DEPARTED).count();
437    double z_ij_sum;
438    double denominator = (n - 1) * (n - 2); // extra 1 subtracted for self
439    double ind;
440
441    if (initial) {
442
443     // first, determine list of z_ij for each agent pair
444     ind = 0; // initial internal network density
445
446     for (Agent agent1 : agents) {
447
448      // calculate z_ij
449      z_ij = 0;
450
451      for (Agent agent2 : agents) {
452       // for now, assume symmetric connection
453       if (agent2.agentIntDensityTable.containsKey(agent1)) {
454        double symmetricValue = agent2.agentIntDensityTable.get(agent1);
455        agent1.agentIntDensityTable.put(agent2, symmetricValue);
456       } else {
457        z_ij = getAgentConnection(agent1, agent2);
458        agent1.agentIntDensityTable.put(agent2, z_ij);
459       }
460
461       // test print
462       // System.out.println(agent1.agentIntDensityTable.get(agent2));
463      }
464      agent1.refreshAveIntNetDensity();
465     }
466    }
467
468    // Calculate average internal network density;
469    // Update values per representation category
470    double num = 0; // overall IND
471    double den = 0; // remaining agents on staff
472
473    double num_low = 0; // low-rep social group
474    double den_low = 0;
475    double num_mod = 0; // mod-rep social group
476    double den_mod = 0;
477    double num_high = 0; // high-rep social group
478    double den_high = 0;
479
480    for (Agent agent : agents) {
481
482     if (agent.mode.get() != AgentMode.DEPARTED) {
```

```
483
484     if (agent.lowRepresentation.get()) {
485      num_low += agent.aveIntDensity;
486      den_low += 1;
487
488      num += agent.aveIntDensity;
489      den += 1;
490
491
492     } else if (agent.moderateRepresentation.get()) {
493      num_mod += agent.aveIntDensity;
494      den_mod += 1;
495
496      num += agent.aveIntDensity;
497      den += 1;
498
499     } else {
500      num_high += agent.aveIntDensity;
501      den_high += 1;
502
503      num += agent.aveIntDensity;
504      den += 1;
505     }
506    }
507   }
508
509   if (den_low == 0) {
510    setTeamIntNetDensityLow(0);
511   } else {
512    setTeamIntNetDensityLow(num_low / den_low);
513   }
514
515   if (den_mod == 0) {
516    setTeamIntNetDensityMod(0);
517   } else {
518    setTeamIntNetDensityMod(num_mod / den_mod);
519   }
520
521   if (den_high == 0) {
522    setTeamIntNetDensityHigh(0);
523   } else {
524    setTeamIntNetDensityHigh(num_high / den_high);
525   }
526
527   if (den == 0) {
528    ind = 0;
529   } else {
530    ind = num / den;
531   }
532
533   // Test print
534   // System.out.println("Team IND: " + ind);
535   return new SimpleDoubleProperty(ind);
536  }
```

```
537
538 public void addAgent(Agent agent) {
539  agents.add(agent);
540  // agent.internalAssignedDemands = this.globalAssignedDemands;
541 }
542
543 class AgentTimeComparator implements Comparator<Agent> {
544
545  @Override
546  public int compare(Agent agent1, Agent agent2) {
547   return Integer.compare(agent1.agentTime.get(), agent2.agentTime.get()
      );
548  }
549
550 }
551
552 // COUNTING CONVENIENCE METHODS
553 public boolean checkAnyDemandList(ListProperty<DemandFX> _demandList) {
554
555  // brute force method to only check initial demands. TODO: adjust if
     initial
556  // demands change
557  for (DemandFX d : _demandList) {
558   // if (d.getType() == ActType.WORK || d.getType() == ActType.DESIGN)
     {
559    if (d.stateProperty().get() != DemandState.COMPLETE) {
560     return true;
561     // }
562
563   }
564  }
565  return false;
566 }
567
568 public long countStateDemands(DemandState _state) {
569  return this.globalAssignedDemands.stream().filter(state -> state.
     stateProperty().get() == _state).count();
570 }
571
572 public long countTypeDemandsRemaining(ActType _type) {
573  return this.globalAssignedDemands.stream().filter(demand -> demand.
     getType() == _type)
574    .filter(d -> d.stateProperty().get() != DemandState.COMPLETE).count
     ();
575 }
576
577 public int countDiversityConsequences() {
578  int count = 0;
579  for (Agent agent : agents) {
580   count += agent.diversityConsequences.size();
581  }
582  return count; // this.diversityConsequences.size(); // was return
     count
583 }
```

```
584
585 private int countLeaderInterventions() {
586  int count = 0;
587  for (Agent agent : agents) {
588   if (agent.getType() == AgentType.LEADER) {
589    count = agent.numInterventions.get();
590   }
591  }
592  return count;
593 }
594
595 private int countLeaderMicroSupports() {
596  int count = 0;
597  for (Agent agent : agents) {
598   if (agent.getType() == AgentType.LEADER) {
599    count = agent.numMicroSupports.get();
600   }
601  }
602  return count;
603 }
604
605 private long countLeadershipTasks() {
606  long count = 0;
607  for (Agent agent : agents) {
608   if (agent.getType() == AgentType.LEADER) {
609    count = agent.internalDemandsMoot.stream().filter(d -> d.getType()
     == ActType.LEAD).count();
610   }
611  }
612  return count;
613 }
614
615 private int countCoordRequests(String _representation) {
616  int count = 0;
617
618  switch (_representation) {
619  case "low":
620   for (Agent agent : agents) {
621    if (agent.lowRepresentation.get()) {
622     count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
     == ActType.COORD)
623        .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
    count();
624    }
625   }
626   for (Agent agent : departedAgents) {
627    if (agent.lowRepresentation.get()) {
628     count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
     == ActType.COORD)
629        .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
    count();
630    }
631   }
632   return count;
```

```
633
634   case "moderate":
635    for (Agent agent : agents) {
636     if (agent.moderateRepresentation.get()) {
637      count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
       == ActType.COORD)
638         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
639     }
640    }
641    for (Agent agent : departedAgents) {
642     if (agent.moderateRepresentation.get()) {
643      count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
       == ActType.COORD)
644         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
645     }
646    }
647    return count;
648   case "high":
649    for (Agent agent : agents) {
650     if (!agent.lowRepresentation.get() || !agent.moderateRepresentation.
      get()) {
651      count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
       == ActType.COORD)
652         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
653     }
654    }
655    for (Agent agent : departedAgents) {
656     if (!agent.lowRepresentation.get() || !agent.moderateRepresentation.
      get()) {
657      count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
       == ActType.COORD)
658         .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
659     }
660    }
661    return count;
662   default:
663    for (Agent agent : agents) {
664     count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
      == ActType.COORD)
665        .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
666    }
667
668    for (Agent agent : departedAgents) {
669     count += agent.internalDemandsMoot.stream().filter(d -> d.getType()
      == ActType.COORD)
670        .filter(d -> d.stateProperty().get() == DemandState.COMPLETE).
      count();
671    }
672    return count;
```

```
673
674   }
675
676  }
677
678  private double countAgents(String representation) {
679
680   switch (representation) {
681   case "low":
682    return agents.stream()
683      .filter(agent -> agent.lowRepresentation.get() == true && agent.
       mode.get() != AgentMode.DEPARTED)
684      .count();
685   // return agents.stream().filter(agent -> agent.lowRepresentation.get
       () ==
686   // true).count();
687   case "moderate":
688    return agents.stream().filter(
689      agent -> agent.moderateRepresentation.get() == true && agent.mode.
       get() != AgentMode.DEPARTED)
690      .count();
691   case "high":
692    return agents.stream().filter(agent -> agent.moderateRepresentation.
       get() != true
693      && agent.lowRepresentation.get() != true && agent.mode.get() !=
       AgentMode.DEPARTED).count();
694   }
695   return agents.size();
696  }
697
698  public String getDiversityConsequencesTrend() {
699   int last_index = diversityConcerns_timeSeries.size() - 1;
700   int pen_index = diversityConcerns_timeSeries.size() - 2;
701
702   Number lastValue = diversityConcerns_timeSeries.get(last_index).
       getYValue();
703   Number penValue = diversityConcerns_timeSeries.get(pen_index).
       getYValue();
704
705   if (lastValue.intValue() - penValue.intValue() < 0) {
706    return "negative";
707   } else if (lastValue.intValue() - penValue.intValue() == 0) {
708    return "flat";
709   }
710   return "positive";
711  }
712
713  public int countAllAbsences() {
714   int count = 0;
715
716   for (Agent agent : agents) {
717    count += agent.getAgentAbsences();
718   }
719
```

```
720   for (Agent agent : departedAgents) {
721    count += agent.getAgentAbsences();
722   }
723   return count;
724  }
725
726  // GETTERS AND SETTERS
727  public StringProperty getName() {
728   return name;
729  }
730
731  public void setName(StringProperty name) {
732   this.name = name;
733  }
734
735  public ObjectProperty<DiversityApproach> getDiversityApproach() {
736   return diversityApproach;
737  }
738
739  public void setDiversityApproach(ObjectProperty<DiversityApproach>
       diversityApproach) {
740   this.diversityApproach = diversityApproach;
741  }
742
743  public ListProperty<DemandFX> getDiversityConcerns() {
744   return diversityConsequences;
745  }
746
747  public void setDiversityConcerns(ListProperty<DemandFX>
       diversityConcerns) {
748   this.diversityConsequences = diversityConcerns;
749  }
750
751  public ListProperty<Agent> getAgents() {
752   return agents;
753  }
754
755  public void setAgents(ListProperty<Agent> agents) {
756   this.agents = agents;
757  }
758
759  public final SimpleListProperty<Data<Number, Number>>
       completedDemands_timeSeriesProperty() {
760   return this.completedDemands_timeSeries;
761  }
762
763  public final ObservableList<Data<Number, Number>>
       getCompletedDemands_timeSeries() {
764   return this.completedDemands_timeSeriesProperty().get();
765  }
766
767  public final void setCompletedDemands_timeSeries(
768    final ObservableList<Data<Number, Number>>
       completedDemands_timeSeries) {
```

```
769   this.completedDemands_timeSeriesProperty().set(
        completedDemands_timeSeries);
770  }
771
772  public final SimpleListProperty<Data<Number, Number>>
       diversityConcerns_timeSeriesProperty() {
773   return this.diversityConcerns_timeSeries;
774  }
775
776  public final ObservableList<Data<Number, Number>>
       getDiversityConcerns_timeSeries() {
777   return this.diversityConcerns_timeSeriesProperty().get();
778  }
779
780  public final void setDiversityConcerns_timeSeries(
781    final ObservableList<Data<Number, Number>>
       diversityConcerns_timeSeries) {
782   this.diversityConcerns_timeSeriesProperty().set(
        diversityConcerns_timeSeries);
783  }
784
785  public final IntegerProperty projectSpanProperty() {
786   return this.projectSpan;
787  }
788
789  public final IntegerProperty getProjectSpan() {
790   return projectSpan;
791  }
792
793  public final void setProjectSpan(final IntegerProperty projectSpan) {
794   this.projectSpan = projectSpan;
795  }
796
797  public final IntegerProperty agentTimeStepProperty() {
798   return this.agentTimeStep;
799  }
800
801  public final int getAgentTimeStep() {
802   return this.agentTimeStepProperty().get();
803  }
804
805  public final void setAgentTimeStep(final int agentTimeStep) {
806   this.agentTimeStepProperty().set(agentTimeStep);
807  }
808
809  public final IntegerProperty n_completeDemandsProperty() {
810   return this.n_completeDemands;
811  }
812
813  public final int getN_completeDemands() {
814   return this.n_completeDemandsProperty().get();
815  }
816
817  public final void setN_completeDemands(final int n_completeDemands) {
```

```
818  this.n_completeDemandsProperty().set(n_completeDemands);
819 }
820
821 public final IntegerProperty globalTimeProperty() {
822  return this.globalTime;
823 }
824
825 public final int getGlobalTime() {
826  return this.globalTimeProperty().get();
827 }
828
829 public final void setGlobalTime(final int globalTime) {
830  this.globalTimeProperty().set(globalTime);
831 }
832
833 public final ListProperty<DemandFX> diversityConsequencesProperty() {
834  return this.diversityConsequences;
835 }
836
837 public final ObservableList<DemandFX> getDiversityConsequences() {
838  return this.diversityConsequencesProperty().get();
839 }
840
841 public final void setDiversityConsequences(final ObservableList<
      DemandFX> diversityConsequences) {
842  this.diversityConsequencesProperty().set(diversityConsequences);
843 }
844
845 public final DoubleProperty teamIntNetDensityProperty() {
846  return this.teamIntNetDensity;
847 }
848
849 public final double getTeamIntNetDensity() {
850  return this.teamIntNetDensityProperty().get();
851 }
852
853 public final void setTeamIntNetDensity(final double teamIntNetDensity)
      {
854  this.teamIntNetDensityProperty().set(teamIntNetDensity);
855 }
856
857 public final SimpleListProperty<Data<Number, Number>>
      intNetDensity_timeSeriesProperty() {
858  return this.intNetDensity_timeSeries;
859 }
860
861 public final ObservableList<Data<Number, Number>>
      getIntNetDensity_timeSeries() {
862  return this.intNetDensity_timeSeriesProperty().get();
863 }
864
865 public final void setIntNetDensity_timeSeries(final ObservableList<Data
      <Number, Number>> intNetDensity_timeSeries) {
866  this.intNetDensity_timeSeriesProperty().set(intNetDensity_timeSeries);
```

```
867 }
868
869 public final SimpleListProperty<Data<Number, Number>>
      lowRepAgentsRetained_timeSeriesProperty() {
870  return this.lowRepAgentsRetained_timeSeries;
871 }
872
873 public final ObservableList<Data<Number, Number>>
      getLowRepAgentsRetained_timeSeries() {
874  return this.lowRepAgentsRetained_timeSeriesProperty().get();
875 }
876
877 public final void setLowRepAgentsRetained_timeSeries(
878   final ObservableList<Data<Number, Number>>
      lowRepAgentsRetained_timeSeries) {
879  this.lowRepAgentsRetained_timeSeriesProperty().set(
      lowRepAgentsRetained_timeSeries);
880 }
881
882 public final SimpleListProperty<Data<Number, Number>>
      modRepAgentsRetained_timeSeriesProperty() {
883  return this.modRepAgentsRetained_timeSeries;
884 }
885
886 public final ObservableList<Data<Number, Number>>
      getModRepAgentsRetained_timeSeries() {
887  return this.modRepAgentsRetained_timeSeriesProperty().get();
888 }
889
890 public final void setModRepAgentsRetained_timeSeries(
891   final ObservableList<Data<Number, Number>>
      modRepAgentsRetained_timeSeries) {
892  this.modRepAgentsRetained_timeSeriesProperty().set(
      modRepAgentsRetained_timeSeries);
893 }
894
895 public final SimpleListProperty<Data<Number, Number>>
      highRepAgentsRetained_timeSeriesProperty() {
896  return this.highRepAgentsRetained_timeSeries;
897 }
898
899 public final ObservableList<Data<Number, Number>>
      getHighRepAgentsRetained_timeSeries() {
900  return this.highRepAgentsRetained_timeSeriesProperty().get();
901 }
902
903 public final void setHighRepAgentsRetained_timeSeries(
904   final ObservableList<Data<Number, Number>>
      highRepAgentsRetained_timeSeries) {
905  this.highRepAgentsRetained_timeSeriesProperty().set(
      highRepAgentsRetained_timeSeries);
906 }
907
908 public final SimpleListProperty<Data<Number, Number>>
```

```
    agentsAbsent_timeSeriesProperty() {
909  return this.agentsAbsent_timeSeries;
910 }
911
912 public final ObservableList<Data<Number, Number>>
      getAgentsAbsent_timeSeries() {
913  return this.agentsAbsent_timeSeriesProperty().get();
914 }
915
916 public final void setAgentsAbsent_timeSeries(final ObservableList<Data<
      Number, Number>> agentsAbsent_timeSeries) {
917  this.agentsAbsent_timeSeriesProperty().set(agentsAbsent_timeSeries);
918 }
919
920 public final ListProperty<DemandFX> globalAssignedDemandsProperty() {
921  return this.globalAssignedDemands;
922 }
923
924 public final ObservableList<DemandFX> getGlobalAssignedDemands() {
925  return this.globalAssignedDemandsProperty().get();
926 }
927
928 public final void setGlobalAssignedDemands(final ObservableList<
      DemandFX> globalAssignedDemands) {
929  this.globalAssignedDemandsProperty().set(globalAssignedDemands);
930 }
931
932 public final ListProperty<DemandFX> globalEmergingSignalsProperty() {
933  return this.globalEmergingSignals;
934 }
935
936 public final ObservableList<DemandFX> getGlobalEmergingSignals() {
937  return this.globalEmergingSignalsProperty().get();
938 }
939
940 public final void setGlobalEmergingSignals(final ObservableList<
      DemandFX> globalEmergingSignals) {
941  this.globalEmergingSignalsProperty().set(globalEmergingSignals);
942 }
943
944 public final ListProperty<DemandFX> globalCoordRequestsProperty() {
945  return this.globalCoordRequests;
946 }
947
948 public final ObservableList<DemandFX> getGlobalCoordRequests() {
949  return this.globalCoordRequestsProperty().get();
950 }
951
952 public final void setGlobalCoordRequests(final ObservableList<DemandFX>
       globalCoordRequests) {
953  this.globalCoordRequestsProperty().set(globalCoordRequests);
954 }
955
956 public final SimpleListProperty<Data<Number, Number>>
```

```
    coordRequests_timeSeriesProperty() {
957  return this.coordRequests_timeSeries;
958 }
959
960 public final ObservableList<Data<Number, Number>>
      getCoordinatedDemands_timeSeries() {
961  return this.coordRequests_timeSeriesProperty().get();
962 }
963
964 public final void setCoordinatedDemands_timeSeries(
965   final ObservableList<Data<Number, Number>>
      coordinatedDemands_timeSeries) {
966  this.coordRequests_timeSeriesProperty().set(
      coordinatedDemands_timeSeries);
967 }
968
969 public final IntegerProperty globalTimeDaysProperty() {
970  return this.globalTimeDays;
971 }
972
973 public final int getGlobalTimeDays() {
974  return this.globalTimeDaysProperty().get();
975 }
976
977 public final void setGlobalTimeDays(final int globalTimeDays) {
978  this.globalTimeDaysProperty().set(globalTimeDays / 28800);
979 }
980
981 public final SimpleListProperty<Data<Number, Number>>
      leadershipTasksComplete_timeSeriesProperty() {
982  return this.leadershipTasks_timeSeries;
983 }
984
985 public final ObservableList<Data<Number, Number>>
      getLeadershipTasksComplete_timeSeries() {
986  return this.leadershipTasksComplete_timeSeriesProperty().get();
987 }
988
989 public final void setLeadershipTasksComplete_timeSeries(
990   final ObservableList<Data<Number, Number>>
      leadershipTasksComplete_timeSeries) {
991  this.leadershipTasksComplete_timeSeriesProperty().set(
      leadershipTasksComplete_timeSeries);
992 }
993
994 public final DoubleProperty n_completeDiversityDemandsProperty() {
995  return this.n_completeDiversityDemands;
996 }
997
998 public final double getN_completeDiversityDemands() {
999  return this.n_completeDiversityDemandsProperty().get();
1000 }
1001
1002 public final void setN_completeDiversityDemands(final double
```

```
    n_completeDiversityDemands) {
1003  this.n_completeDiversityDemandsProperty().set(
    n_completeDiversityDemands);
1004 }
1005
1006 public final ListProperty<Agent> departedAgentsProperty() {
1007  return this.departedAgents;
1008 }
1009
1010 public final ObservableList<Agent> getDepartedAgents() {
1011  return this.departedAgentsProperty().get();
1012 }
1013
1014 public final void setDepartedAgents(final ObservableList<Agent>
    departedAgents) {
1015  this.departedAgentsProperty().set(departedAgents);
1016 }
1017
1018 public final DoubleProperty n_agentsLost_lowProperty() {
1019  return this.n_agentsLost_low;
1020 }
1021
1022 public final double getN_agentsLost_low() {
1023  return this.n_agentsLost_lowProperty().get();
1024 }
1025
1026 public final void setN_agentsLost_low(final double n_agentsLost_low) {
1027  this.n_agentsLost_lowProperty().set(n_agentsLost_low);
1028 }
1029
1030 public final DoubleProperty n_agentsLost_moderateProperty() {
1031  return this.n_agentsLost_moderate;
1032 }
1033
1034 public final double getN_agentsLost_moderate() {
1035  return this.n_agentsLost_moderateProperty().get();
1036 }
1037
1038 public final void setN_agentsLost_moderate(final double
    n_agentsLost_moderate) {
1039  this.n_agentsLost_moderateProperty().set(n_agentsLost_moderate);
1040 }
1041
1042 public final DoubleProperty n_agentsLost_highProperty() {
1043  return this.n_agentsLost_high;
1044 }
1045
1046 public final double getN_agentsLost_high() {
1047  return this.n_agentsLost_highProperty().get();
1048 }
1049
1050 public final void setN_agentsLost_high(final double n_agentsLost_high)
    {
1051  this.n_agentsLost_highProperty().set(n_agentsLost_high);
```

```
1052  }
1053
1054  public final DoubleProperty indStartProperty() {
1055   return this.indStart;
1056  }
1057
1058  public final double getIndStart() {
1059   return this.indStartProperty().get();
1060  }
1061
1062  public final void setIndStart(final double indStart) {
1063   this.indStartProperty().set(indStart);
1064  }
1065
1066  public final DoubleProperty indFinishProperty() {
1067   return this.indFinish;
1068  }
1069
1070  public final double getIndFinish() {
1071   return this.indFinishProperty().get();
1072  }
1073
1074  public final void setIndFinish(final double indFinish) {
1075   this.indFinishProperty().set(indFinish);
1076  }
1077
1078  public final DoubleProperty teamIntNetDensityLowProperty() {
1079   return this.teamIntNetDensityLow;
1080  }
1081
1082  public final double getTeamIntNetDensityLow() {
1083   return this.teamIntNetDensityLowProperty().get();
1084  }
1085
1086  public final void setTeamIntNetDensityLow(final double
        teamIntNetDensityLow) {
1087   this.teamIntNetDensityLowProperty().set(teamIntNetDensityLow);
1088  }
1089
1090  public final DoubleProperty teamIntNetDensityModProperty() {
1091   return this.teamIntNetDensityMod;
1092  }
1093
1094  public final double getTeamIntNetDensityMod() {
1095   return this.teamIntNetDensityModProperty().get();
1096  }
1097
1098  public final void setTeamIntNetDensityMod(final double
        teamIntNetDensityMod) {
1099   this.teamIntNetDensityModProperty().set(teamIntNetDensityMod);
1100  }
1101
1102  public final DoubleProperty teamIntNetDensityHighProperty() {
1103   return this.teamIntNetDensityHigh;
```

```
1104 }
1105
1106 public final double getTeamIntNetDensityHigh() {
1107  return this.teamIntNetDensityHighProperty().get();
1108 }
1109
1110 public final void setTeamIntNetDensityHigh(final double
       teamIntNetDensityHigh) {
1111  this.teamIntNetDensityHighProperty().set(teamIntNetDensityHigh);
1112 }
1113
1114 public final SimpleListProperty<Data<Number, Number>>
       intNetDensityLow_timeSeriesProperty() {
1115  return this.intNetDensityLow_timeSeries;
1116 }
1117
1118 public final ObservableList<Data<Number, Number>>
       getIntNetDensityLow_timeSeries() {
1119  return this.intNetDensityLow_timeSeriesProperty().get();
1120 }
1121
1122 public final void setIntNetDensityLow_timeSeries(
1123   final ObservableList<Data<Number, Number>>
       intNetDensityLow_timeSeries) {
1124  this.intNetDensityLow_timeSeriesProperty().set(
       intNetDensityLow_timeSeries);
1125 }
1126
1127 public final SimpleListProperty<Data<Number, Number>>
       intNetDensityMod_timeSeriesProperty() {
1128  return this.intNetDensityMod_timeSeries;
1129 }
1130
1131 public final ObservableList<Data<Number, Number>>
       getIntNetDensityMod_timeSeries() {
1132  return this.intNetDensityMod_timeSeriesProperty().get();
1133 }
1134
1135 public final void setIntNetDensityMod_timeSeries(
1136   final ObservableList<Data<Number, Number>>
       intNetDensityMod_timeSeries) {
1137  this.intNetDensityMod_timeSeriesProperty().set(
       intNetDensityMod_timeSeries);
1138 }
1139
1140 public final SimpleListProperty<Data<Number, Number>>
       intNetDensityHigh_timeSeriesProperty() {
1141  return this.intNetDensityHigh_timeSeries;
1142 }
1143
1144 public final ObservableList<Data<Number, Number>>
       getIntNetDensityHigh_timeSeries() {
1145  return this.intNetDensityHigh_timeSeriesProperty().get();
1146 }
```

```
1147
1148 public final void setIntNetDensityHigh_timeSeries(
1149   final ObservableList<Data<Number, Number>>
       intNetDensityHigh_timeSeries) {
1150  this.intNetDensityHigh_timeSeriesProperty().set(
       intNetDensityHigh_timeSeries);
1151 }
1152
1153 public final DoubleProperty n_diversityDemandsProperty() {
1154  return this.n_diversityDemands;
1155 }
1156
1157 public final double getN_diversityDemands() {
1158  return this.n_diversityDemandsProperty().get();
1159 }
1160
1161 public final void setN_diversityDemands(final double n_diversityDemands
       ) {
1162  this.n_diversityDemandsProperty().set(n_diversityDemands);
1163 }
1164
1165 public final IntegerProperty n_agentAbsencesProperty() {
1166  return this.n_agentAbsences;
1167 }
1168
1169 public final int getN_agentAbsences() {
1170  return this.n_agentAbsencesProperty().get();
1171 }
1172
1173 public final void setN_agentAbsences(final int n_agentAbsences) {
1174  this.n_agentAbsencesProperty().set(n_agentAbsences);
1175 }
1176
1177 public final IntegerProperty n_leaderInterventionsProperty() {
1178  return this.n_leaderInterventions;
1179 }
1180
1181 public final int getN_leaderInterventions() {
1182  return this.n_leaderInterventionsProperty().get();
1183 }
1184
1185 public final void setN_leaderInterventions(final int
       n_leaderInterventions) {
1186  this.n_leaderInterventionsProperty().set(n_leaderInterventions);
1187 }
1188
1189 public final IntegerProperty n_leaderMicroSupportsProperty() {
1190  return this.n_leaderMicroSupports;
1191 }
1192
1193 public final int getN_leaderMicroSupports() {
1194  return this.n_leaderMicroSupportsProperty().get();
1195 }
1196
```

```
1197  public final void setN_leaderMicroSupports(final int
        n_leaderMicroSupports) {
1198   this.n_leaderMicroSupportsProperty().set(n_leaderMicroSupports);
1199  }
1200
1201  public final IntegerProperty n_interactionsProperty() {
1202   return this.n_interactions;
1203  }
1204
1205  public final int getN_interactions() {
1206   return this.n_interactionsProperty().get();
1207  }
1208
1209  public final void setN_interactions(final int n_interactions) {
1210   this.n_interactionsProperty().set(n_interactions);
1211  }
1212
1213  public final IntegerProperty n_interactionsLowProperty() {
1214   return this.n_interactionsLow;
1215  }
1216
1217  public final int getN_interactionsLow() {
1218   return this.n_interactionsLowProperty().get();
1219  }
1220
1221  public final void setN_interactionsLow(final int n_interactionsLow) {
1222   this.n_interactionsLowProperty().set(n_interactionsLow);
1223  }
1224
1225  public final IntegerProperty n_interactionsModProperty() {
1226   return this.n_interactionsMod;
1227  }
1228
1229  public final int getN_interactionsMod() {
1230   return this.n_interactionsModProperty().get();
1231  }
1232
1233  public final void setN_interactionsMod(final int n_interactionsMod) {
1234   this.n_interactionsModProperty().set(n_interactionsMod);
1235  }
1236
1237  public final IntegerProperty n_interactionsHighProperty() {
1238   return this.n_interactionsHigh;
1239  }
1240
1241  public final int getN_interactionsHigh() {
1242   return this.n_interactionsHighProperty().get();
1243  }
1244
1245  public final void setN_interactionsHigh(final int n_interactionsHigh) {
1246   this.n_interactionsHighProperty().set(n_interactionsHigh);
1247  }
1248
1249 }
```

A.2.4 DiversityApproach.java

```
import java.util.Random;

public enum DiversityApproach {
 EQUALITY, DIFFERENCE;

 public static DiversityApproach getRandomDiversityApproach() {
  Random random = new Random();
  if (random.nextInt(10) <= 1) {
   return values()[0];
  } else {
   return values()[1];
  }

 }
}
```

A.2.5 DemandBase.java

```

import java.util.Random;
import java.util.UUID;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class DemandBase implements DemandFX {

 private UUID id;
 private StringProperty name; // a human readable short name for use UI
    labels
 private ObjectProperty<ActType> type;
 private ObjectProperty<DemandState> state;
 private ObjectProperty<Agent> requestor;
 private ObjectProperty<Agent> helper;

 private IntegerProperty effort; // effort in seconds
 private IntegerProperty priority;
 private IntegerProperty start;
 private IntegerProperty stop;

 // private BooleanProperty recur;
 // private IntegerProperty every;
 // private IntegerProperty until;

 public BooleanProperty coordNeeded; // parameter to drive coordination

 // DEFAULT VALUES
```

```
34 private static ActType DEFAULT_TYPE = ActType.DESIGN;
35 private static DemandState DEFAULT_STATE = DemandState.INACTIVE;
36 private static int DEFAULT_START = 9;
37 private static int DEFAULT_STOP = 17;
38 private static int DEFAULT_EFFORT = 3600;
39
40
41 // private static int DEFAULT_EVERY = 24;
42 // private static int DEFAULT_UNTIL = 24 * 14;
43 // private static boolean DEFAULT_RECUR = false;
44
45 /**
46  * Constructs a demand with default state INACTIVE with default values
47  */
48 public DemandBase() {
49  this(DEFAULT_TYPE, DEFAULT_EFFORT);
50 }
51
52 /**
53  * Constructs a demand with default state INACTIVE with _type and
      nominal
54  * _effortSecs
55  *
56  * @param _type       the type of demand effort from set of TYPE_
      constants in
57  *                    this class
58  * @param _effortSecs in seconds
59  */
60 public DemandBase(ActType _type, int _effortSecs) {
61  this(_type, _effortSecs, DEFAULT_START, DEFAULT_STOP);
62 }
63
64 /**
65  * Constructs a demand object with default name and the following input
       values
66  *
67  * @param _type       the type of demand effort from set of TYPE_
      constants in
68  *                    this class
69  * @param _effortSecs nominal effort in seconds
70  * @param _start      the start time as int of this demand
71  * @param _stop       the stop time as int of this demand
72  */
73 public DemandBase(ActType _type, int _effortSecs, int _start, int _stop
      ) {
74  // default name is type and effort in hours
75  this(_type, _effortSecs, _start, _stop, _type + ": " + _effortSecs /
      3600.0);
76 }
77
78 /**
79  * Constructs a demand object with defaults and the following input
      values
80  *
```

```
81   * @param _type       the type of demand effort from set of TYPE_
       constants in
82   *                    this class
83   * @param _effortSecs nominal effort in seconds
84   * @param _start      the start time as int of this demand
85   * @param _stop       the stop time as int of this demand
86   * @param _name       a short text name for human readible tag of this
       demand
87   */
88  public DemandBase(ActType _type, int _effortSecs, int _start, int _stop
       , String _name) {
89   id = UUID.randomUUID();
90   type = new SimpleObjectProperty<>(_type);
91   state = new SimpleObjectProperty<>(DEFAULT_STATE);
92   requestor = new SimpleObjectProperty<Agent>();
93
94   effort = new SimpleIntegerProperty(_effortSecs);
95   start = new SimpleIntegerProperty(_start);
96   stop = new SimpleIntegerProperty(_stop);
97   name = new SimpleStringProperty(_name);
98   priority = new SimpleIntegerProperty(2);
99   // coordNeeded = new SimpleBooleanProperty(RandomCoordNeeded());
100 //   recur = new SimpleBooleanProperty(DEFAULT_RECUR);
101 //   every = new SimpleIntegerProperty(DEFAULT_EVERY);
102 //   until = new SimpleIntegerProperty(DEFAULT_UNTIL);
103  }
104
105  // Convenience Methods
106  private boolean RandomCoordNeeded() {
107   Random rand = new Random();
108
109   if (this.getType() != ActType.WORK) {
110    if (rand.nextInt(100) < 25) { // ASSUMPTION: 25% task interdependence
111     return true;
112    }
113   }
114   return false;
115  }
116
117  @Override
118  public String toString() {
119   String retVal = "Demand - " + name.get() + " [" + start.get() + "-" +
       stop.get() + "] " + type.get() + ": "
120    + getEffortHrs() + " hrs";
121   // if (recur.get()) {
122   // retVal += " every " + every.get() + " until " + stop.get();
123   // }
124   return retVal;
125  }
126
127  /**
128   * Returns true of the demand is active during the period from start to
       top
129   *
```

```
130   * @param now an in representing time
131   * @return true if the now time is during an active period for this
        demand
132   */
133  public boolean isActive(int now) {
134   return (now >= start.get() && now <= stop.get());
135  }
136
137  // BELOW is a BUNCH of boilerplate code for getting, setting, and
       accessing
138  // properties
139  // Yes, it is verbose, one of the challenges of Java. In most cases the
        code can
140  // be automatically generated.
141  // While it would be MUCH simpler to just have a public property which
       is then
142  // .get() and .set(x),
143  // It is best practice and convention in Java to encapsulate internal
       variables
144  // and to conform to
145  // convention and backward compatibility by have the get and set calls.
        Take a
146  // look at the alternative code which is much shorter,
147  // but which would lead to less flexibility and compatibility in later
148  // development
149
150  @Override
151  public UUID getId() {
152   return id;
153  }
154
155  @Override
156  public void setId(UUID id) {
157   this.id = id;
158  }
159
160  // NAME ------------------------------------
161  @Override
162  public StringProperty nameProperty() {
163   return name;
164  }
165  @Override
166  public String getName() {
167   return name.get();
168  }
169
170  @Override
171  public void setName(String name) {
172   this.name.set(name);
173  }
174
175  // PRIORITY --------------------------------
176  @Override
177  public IntegerProperty priorityProperty() {
```

```
178   return priority;
179  }
180
181  @Override
182  public int getPriority() {
183   return priority.get();
184  }
185
186  @Override
187  public void setPriority(int priority) {
188   this.priority.set(priority);
189  }
190
191  // TYPE ----------------------------------
192  @Override
193  public ObjectProperty<ActType> typeProperty() {
194   return type;
195  }
196
197  @Override
198  public ActType getType() {
199   return type.get();
200  }
201
202  @Override
203  public void setType(ActType type) {
204   this.type.set(type);
205  }
206
207  // STATE ---------------------------------
208  @Override
209  public ObjectProperty<DemandState> stateProperty() {
210   return state;
211  }
212
213  @Override
214  public DemandState getState() {
215   return state.get();
216  }
217
218  @Override
219  public void setState(DemandState state) {
220   this.state.set(state);
221  }
222
223  // EFFORT --------------------------------
224  @Override
225  public IntegerProperty effortProperty() {
226   return effort;
227  }
228
229  /*
230   * Returns the effort of this demand in seconds
231   */
```

```
232 @Override
233 public int getEffort() {
234  return effort.get();
235 }
236
237 @Override
238 public void setEffort(int nominalEffort_seconds) {
239  this.effort.set(nominalEffort_seconds);
240 }
241
242 /*
243  * Returns the effort of this demand in hours (convenience method)
244  */
245 @Override
246 public double getEffortHrs() {
247  return effort.get() / 3600.0;
248 }
249
250 @Override
251 public void setEffortHrs(double _effort) {
252  this.effort.set((int) (_effort * 3600));
253 }
254
255 // START --------------------------------------
256 @Override
257 public IntegerProperty startProperty() {
258  return start;
259 }
260 @Override
261 public int getStart() {
262  return start.get();
263 }
264 @Override
265 public void setStart(int start) {
266  this.start.set(start);
267 }
268
269 // STOP --------------------------------------
270 @Override
271 public IntegerProperty stopProperty() {
272  return stop;
273 }
274 @Override
275 public int getStop() {
276  return stop.get();
277 }
278 @Override
279 public void setStop(int stop) {
280  this.stop.set(stop);
281 }
282
283 // COORD --------------------------------------
284 @Override
285 public BooleanProperty coordNeeded() {
```

```
286   return coordNeeded;
287  }
288
289  @Override
290  public boolean getCoordNeeded() {
291   return coordNeeded.get();
292  }
293
294  @Override
295  public void setCoordNeeded(boolean coord) {
296   this.coordNeeded.set(coord);
297  }
298
299  // RECUR ---------------------------------
300 // @Override
301 // public BooleanProperty recurProperty() {
302 //  return recur;
303 // }
304 //
305 // @Override
306 // public boolean getRecur() {
307 //  return recur.get();
308 // }
309 //
310 // @Override
311 // public void setRecur(boolean isRecurring) {
312 //  this.recur.set(isRecurring);
313 // }
314 // @Override
315 // public boolean isRecur() {
316 //  return recur.get();
317 // }
318 //
319 // // EVERY ---------------------------------
320 // @Override
321 // public IntegerProperty everyProperty() {
322 //  return every;
323 // }
324 // @Override
325 // public int getEvery() {
326 //  return every.get();
327 // }
328 // @Override
329 // public void setEvery(int every) {
330 //  this.every.set(every);
331 // }
332 //
333 // // UNTIL ---------------------------------
334 // @Override
335 // public IntegerProperty untilProperty() {
336 //  return until;
337 // }
338 //
339 // @Override
```

```
340 // public int getUntil() {
341 //  return until.get();
342 // }
343 //
344 // @Override
345 // public void setUntil(int until) {
346 //  this.until.set(until);
347 // }
348 //
349 // // CONVENIENCE METHODS
350 // /**
351 //  * Set the active period of this demand from start to stop does now
      allow stop
352 //  * before start
353 //  *
354 //  * @param _start
355 //  * @param _stop
356 //  */
357 // public void setPeriod(int start, int stop) {
358 //  if (start > stop) {
359 //   return;
360 //  }
361 //  this.start.set(start);
362 //  this.stop.set(stop);
363 // }
364 //
365  @Override
366  public ObjectProperty<Agent> getRequestor() {
367   return requestor;
368  }
369
370  @Override
371  public void setRequestor(ObjectProperty<Agent> requestor) {
372   this.requestor = requestor;
373  }
374
375  @Override
376  public ObjectProperty<Agent> requestorProperty() {
377   return requestor;
378  }
379
380  @Override
381  public ObjectProperty<Agent> getHelper() {
382   return helper;
383  }
384
385  @Override
386  public void setHelper(ObjectProperty<Agent> helper) {
387   this.helper = helper;
388  }
389
390  @Override
391  public ObjectProperty<Agent> helperProperty() {
392   return helper;
```

```
 }
}
```

A.2.6 DemandFX.java

```

import java.util.UUID;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.StringProperty;

public interface DemandFX {

 // A property, from JavaFX, allows the attributes to be observable (and
 // bindable)
 // if only working with a text UI, these attributes could be their
    plain type
 // (e.g. int, String)

 // BELOW is a BUNCH of boilerplate code for getting, setting, and
    accessing
 // properties
 // Yes, it is verbose, one of the challenges of Java. In most cases the
     code can
 // be automatically generated.
 // While it would be MUCH simpler to just have a public property which
    is then
 // .get() and .set(x),
 // It is best practice and convention in Java to encapsulate internal
    variables
 // and to conform to
 // convention and backward compatibility by have the get and set calls.
     Take a
 // look at the alternative code which is much shorter,
 // but which would lead to less flexibility and compatibility in later
 // development

 public UUID getId();

 public void setId(UUID id);

 // NAME ------------------------------------
 public StringProperty nameProperty();

 public String getName();

 public void setName(String name);

 // PRIORITY --------------------------------
 public IntegerProperty priorityProperty();

 public int getPriority();
```

```
44
45  public void setPriority(int priority);
46
47  // TYPE ----------------------------------
48  public ObjectProperty<ActType> typeProperty();
49
50  public ActType getType();
51
52  public void setType(ActType type);
53
54  // REQUESTOR -----------------------------------
55  public ObjectProperty<Agent> requestorProperty();
56
57  public ObjectProperty<Agent> getRequestor();
58
59  public void setRequestor(ObjectProperty<Agent> agent);
60
61  // HELPER --------------------------------------
62  public ObjectProperty<Agent> helperProperty();
63
64  public ObjectProperty<Agent> getHelper();
65
66  public void setHelper(ObjectProperty<Agent> agentFX);
67
68  // STATE ---------------------------------------
69  public ObjectProperty<DemandState> stateProperty();
70
71  public DemandState getState();
72
73  public void setState(DemandState state);
74
75  // EFFORT --------------------------------------
76  public IntegerProperty effortProperty();
77
78  public int getEffort();
79
80  public void setEffort(int nominalEffort_seconds);
81
82  /*
83   * Returns the effort of this demand in hours (convenience method)
84   */
85  public double getEffortHrs();
86
87  public void setEffortHrs(double _effort);
88
89  // START ---------------------------------------
90  public IntegerProperty startProperty();
91
92  public int getStart();
93
94  public void setStart(int start);
95
96  // STOP ----------------------------------------
97  public IntegerProperty stopProperty();
```

```
98
99  public int getStop();
100
101  public void setStop(int stop);
102
103  // COORD-------------------------------------
104  public BooleanProperty coordNeeded();
105
106  public boolean getCoordNeeded();
107
108  public void setCoordNeeded(boolean coord);
109
110 // // RECUR -------------------------------------
111 // public BooleanProperty recurProperty();
112 // public boolean getRecur();
113 // public void setRecur(boolean isRecurring);
114 // public boolean isRecur();
115 //
116 // // EVERY -------------------------------------
117 // public IntegerProperty everyProperty();
118 // public int getEvery();
119 // public void setEvery(int every);
120 //
121 // // UNTIL -------------------------------------
122 // public IntegerProperty untilProperty();
123 // public int getUntil();
124 // public void setUntil(int until);
125 }
```

A.2.7 DemandState.java

```
1
2 public enum DemandState {
3  INACTIVE, PENDING, STARTED, ACTIVE, COMPLETE, IGNORED
4 }
```

A.2.8 DemandType.java

```
1
2 public enum DemandType {
3  COORD, DECIDE, DESIGN, TRAVEL, ADMIN, LEAD
4 }
```

A.2.9 SimResult.java

```
1 import javafx.beans.property.DoubleProperty;
2 import javafx.beans.property.IntegerProperty;
3
4 public class SimResult {
5  IntegerProperty numInteractions;
6  IntegerProperty projectDuration;
7  DoubleProperty diversityDemandsGenerated;
8  DoubleProperty diversityDemandsComplete;
9  DoubleProperty intNetDensityStart;
```

```
10  DoubleProperty intNetDensityFinish;
11  IntegerProperty agentAbsences;
12  DoubleProperty numAgentsLostLow;
13  DoubleProperty numAgentsLostMod;
14  DoubleProperty numAgentsLostHigh;
15  IntegerProperty numLeaderInterventions;
16  IntegerProperty numLeaderMicroSupports;
17
18  public SimResult(World _agentEnvironment) {
19   projectDuration = _agentEnvironment.globalTime;
20   diversityDemandsGenerated = _agentEnvironment.n_diversityDemands;
21   diversityDemandsComplete = _agentEnvironment.
       n_completeDiversityDemands;
22   numInteractions = _agentEnvironment.n_interactions;
23
24   agentAbsences = _agentEnvironment.n_agentAbsences;
25
26   intNetDensityStart = _agentEnvironment.indStart;
27   intNetDensityFinish = _agentEnvironment.indFinish;
28
29   numLeaderInterventions = _agentEnvironment.n_leaderInterventions;
30   numLeaderMicroSupports = _agentEnvironment.n_leaderMicroSupports;
31
32   numAgentsLostLow = _agentEnvironment.n_agentsLost_low;
33   numAgentsLostMod = _agentEnvironment.n_agentsLost_moderate;
34   numAgentsLostHigh = _agentEnvironment.n_agentsLost_high;
35
36   // Screen Output//
37 //   System.out.println("\nProject Duration:    " + projectDuration.get()
       + "\nDiversity Concerns: "
38 //     + diversityDemandsGenerated.get() + "\nInteractions:       " +
       numInteractions.get()
39 //     + "\nAbsences:           " + agentAbsences.get() + "\nIND Start:
             " + intNetDensityStart.get()
40 //     + "\nIND Finish:         " + intNetDensityFinish.get() + "\
       nInterventions:      "
41 //     + numLeaderInterventions.get() + "\nDaily Supports:     " +
       numLeaderMicroSupports.get()
42 //     + "\nAgents Lost (Low):  " + numAgentsLostLow.get() + "\nAgents
       Lost (Mod):  " + numAgentsLostMod.get()
43 //     + "\nAgents Lost (High): " + numAgentsLostHigh.get());
44
45   System.out.println("Data collected");
46
47  }
48
49  public final IntegerProperty projectDurationProperty() {
50   return this.projectDuration;
51  }
52
53  public final int getProjectDuration() {
54   return this.projectDurationProperty().get();
55  }
56
```

```
57 public final void setProjectDuration(final int projectDuration) {
58  this.projectDurationProperty().set(projectDuration);
59 }
60
61 public final DoubleProperty diversityDemandsCompleteProperty() {
62  return this.diversityDemandsComplete;
63 }
64
65 public final double getDiversityDemandsComplete() {
66  return this.diversityDemandsCompleteProperty().get();
67 }
68
69 public final void setDiversityDemandsComplete(final double
     diversityDemandsComplete) {
70  this.diversityDemandsCompleteProperty().set(diversityDemandsComplete);
71 }
72
73 public final DoubleProperty intNetDensityStartProperty() {
74  return this.intNetDensityStart;
75 }
76
77 public final double getIntNetDensityStart() {
78  return this.intNetDensityStartProperty().get();
79 }
80
81 public final void setIntNetDensityStart(final double intNetDensityStart
     ) {
82  this.intNetDensityStartProperty().set(intNetDensityStart);
83 }
84
85 public final DoubleProperty intNetDensityFinishProperty() {
86  return this.intNetDensityFinish;
87 }
88
89 public final double getIntNetDensityFinish() {
90  return this.intNetDensityFinishProperty().get();
91 }
92
93 public final void setIntNetDensityFinish(final double
     intNetDensityFinish) {
94  this.intNetDensityFinishProperty().set(intNetDensityFinish);
95 }
96
97 public final DoubleProperty numAgentsLostLowProperty() {
98  return this.numAgentsLostLow;
99 }
100
101 public final double getNumAgentsLostLow() {
102  return this.numAgentsLostLowProperty().get();
103 }
104
105 public final void setNumAgentsLostLow(final double numAgentsLostLow) {
106  this.numAgentsLostLowProperty().set(numAgentsLostLow);
107 }
```

```
108
109 public final DoubleProperty numAgentsLostModProperty() {
110  return this.numAgentsLostMod;
111 }
112
113 public final double getNumAgentsLostMod() {
114  return this.numAgentsLostModProperty().get();
115 }
116
117 public final void setNumAgentsLostMod(final double numAgentsLostMod) {
118  this.numAgentsLostModProperty().set(numAgentsLostMod);
119 }
120
121 public final DoubleProperty numAgentsLostHighProperty() {
122  return this.numAgentsLostHigh;
123 }
124
125 public final double getNumAgentsLostHigh() {
126  return this.numAgentsLostHighProperty().get();
127 }
128
129 public final void setNumAgentsLostHigh(final double numAgentsLostHigh)
      {
130  this.numAgentsLostHighProperty().set(numAgentsLostHigh);
131 }
132
133 public final IntegerProperty numLeaderInterventionsProperty() {
134  return this.numLeaderInterventions;
135 }
136
137 public final int getNumLeaderInterventions() {
138  return this.numLeaderInterventionsProperty().get();
139 }
140
141 public final void setNumLeaderInterventions(final int
      numLeaderInterventions) {
142  this.numLeaderInterventionsProperty().set(numLeaderInterventions);
143 }
144
145 public final IntegerProperty numLeaderMicroSupportsProperty() {
146  return this.numLeaderMicroSupports;
147 }
148
149 public final int getNumLeaderMicroSupports() {
150  return this.numLeaderMicroSupportsProperty().get();
151 }
152
153 public final void setNumLeaderMicroSupports(final int
      numLeaderMicroSupports) {
154  this.numLeaderMicroSupportsProperty().set(numLeaderMicroSupports);
155 }
156
157 public final DoubleProperty diversityDemandsGeneratedProperty() {
158  return this.diversityDemandsGenerated;
```

```
159  }
160
161  public final double getDiversityDemandsGenerated() {
162   return this.diversityDemandsGeneratedProperty().get();
163  }
164
165  public final void setDiversityDemandsGenerated(final double
       diversityDemandsGenerated) {
166   this.diversityDemandsGeneratedProperty().set(diversityDemandsGenerated
       );
167  }
168
169  public final IntegerProperty agentAbsencesProperty() {
170   return this.agentAbsences;
171  }
172
173  public final int getAgentAbsences() {
174   return this.agentAbsencesProperty().get();
175  }
176
177  public final void setAgentAbsences(final int agentAbsences) {
178   this.agentAbsencesProperty().set(agentAbsences);
179  }
180
181  public final IntegerProperty numCoordinationsProperty() {
182   return this.numInteractions;
183  }
184
185  public final int getNumCoordinations() {
186   return this.numCoordinationsProperty().get();
187  }
188
189  public final void setNumCoordinations(final int numCoordinations) {
190   this.numCoordinationsProperty().set(numCoordinations);
191  }
192
193  public final IntegerProperty numInteractionsProperty() {
194   return this.numInteractions;
195  }
196
197  public final int getNumInteractions() {
198   return this.numInteractionsProperty().get();
199  }
200
201  public final void setNumInteractions(final int numInteractions) {
202   this.numInteractionsProperty().set(numInteractions);
203  }
204
205 }
```

A.2.10 ResultsExporter.java

```
1 import java.io.File;
2 import java.io.FileNotFoundException;
```

```
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.util.List;
6
7  import org.apache.poi.ss.usermodel.BorderStyle;
8  import org.apache.poi.ss.usermodel.Cell;
9  import org.apache.poi.ss.usermodel.CellStyle;
10 import org.apache.poi.ss.usermodel.HorizontalAlignment;
11 import org.apache.poi.ss.usermodel.Row;
12 import org.apache.poi.xssf.usermodel.DefaultIndexedColorMap;
13 import org.apache.poi.xssf.usermodel.XSSFCellStyle;
14 import org.apache.poi.xssf.usermodel.XSSFColor;
15 import org.apache.poi.xssf.usermodel.XSSFFont;
16 import org.apache.poi.xssf.usermodel.XSSFSheet;
17 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
18
19 public class ResultsExporter {
20
21  List<SimResult> results;
22  int count;
23  CellStyle styleBold;
24
25  XSSFCellStyle style, keyStyle;
26  XSSFFont font, keyFont;
27
28  public ResultsExporter() {
29
30  }
31
32  public ResultsExporter(List<SimResult> _results) {
33   // this();
34   results = _results;
35   // count = _results.size();
36  }
37
38  public void export(File fileToWrite) {
39   if (results == null || fileToWrite == null) {
40    return;
41   }
42   FileOutputStream fos = null;
43   try {
44    fos = new FileOutputStream(fileToWrite);
45   } catch (FileNotFoundException e1) {
46    e1.printStackTrace();
47   }
48
49   XSSFWorkbook workbook = new XSSFWorkbook();
50   // common styles
51   style = workbook.createCellStyle();
52   style.setBorderBottom(BorderStyle.THICK);
53   style.setAlignment(HorizontalAlignment.CENTER);
54
55   font = workbook.createFont();
56   font.setFontHeightInPoints((short) 10.5);
```

```
57   font.setBold(true);
58   style.setFont(font);
59
60   keyStyle = workbook.createCellStyle();
61   keyStyle.setBorderBottom(BorderStyle.THIN);
62   keyStyle.setAlignment(HorizontalAlignment.CENTER);
63   keyFont = workbook.createFont();
64   keyFont.setFontHeightInPoints((short) 10.5);
65   keyFont.setBold(true);
66   keyFont.setColor(new XSSFColor(new java.awt.Color(32, 32, 160), new
      DefaultIndexedColorMap()));
67   keyStyle.setFont(keyFont);
68
69   // write sheets
70
71   XSSFSheet sheet = workbook.createSheet("SimResults");
72
73   // sheet = workbook.createSheet("Result Data");
74   writeHeaderLineData(sheet);
75   writeRowsData(results, workbook, sheet);
76   writeStylesData(sheet);
77
78   try {
79    workbook.write(fos);
80    workbook.close();
81   } catch (IOException e) {
82    e.printStackTrace();
83   }
84  }
85
86  private void writeHeaderLineData(XSSFSheet sheet) {
87
88   Row headerRow = sheet.createRow(0);
89
90   Cell headerCell = headerRow.createCell(0);
91   headerCell.setCellValue("Num Interactions");
92
93   headerCell = headerRow.createCell(1);
94   headerCell.setCellValue("Starting Internal Network Density");
95
96   headerCell = headerRow.createCell(2);
97   headerCell.setCellValue("Final Internal Network Density");
98
99   headerCell = headerRow.createCell(3);
100  headerCell.setCellValue("Diversity Concerns Generated");
101
102  headerCell = headerRow.createCell(4);
103  headerCell.setCellValue("Diversity Concerns Addressed");
104
105  headerCell = headerRow.createCell(5);
106  headerCell.setCellValue("Absences");
107
108  headerCell = headerRow.createCell(6);
109  headerCell.setCellValue("Leader Interventions");
```

```
110
111   headerCell = headerRow.createCell(7);
112   headerCell.setCellValue("Leader MicroSupports");
113
114   headerCell = headerRow.createCell(8);
115   headerCell.setCellValue("Agents Lost (Low Rep)");
116
117   headerCell = headerRow.createCell(9);
118   headerCell.setCellValue("Agents Lost (Mod Rep)");
119
120   headerCell = headerRow.createCell(10);
121   headerCell.setCellValue("Agents Lost (High Rep)");
122
123   headerCell = headerRow.createCell(11);
124   headerCell.setCellValue("Project Duration");
125
126  }
127
128  private void writeStylesData(XSSFSheet sheet) {
129
130   sheet.autoSizeColumn(0);
131   sheet.autoSizeColumn(1);
132   sheet.autoSizeColumn(2);
133   sheet.autoSizeColumn(3);
134   sheet.autoSizeColumn(4);
135   sheet.autoSizeColumn(5);
136   sheet.autoSizeColumn(6);
137   sheet.autoSizeColumn(7);
138   sheet.autoSizeColumn(8);
139   sheet.autoSizeColumn(9);
140   sheet.autoSizeColumn(10);
141
142   Row row = sheet.getRow(0);
143   for (int j = 0; j < row.getLastCellNum(); j++) {
144    row.getCell(j).setCellStyle(style);
145   }
146
147  }
148
149  private void writeRowsData(List<SimResult> results, XSSFWorkbook
       workbook, XSSFSheet sheet) {
150
151   int rowCount = 1;
152
153   for (SimResult r : results) {
154
155     Row row = sheet.createRow(rowCount++);
156
157     int columnCount = 0;
158
159     Cell cell = row.createCell(columnCount++);
160     cell.setCellValue(r.getNumInteractions());
161
162     cell = row.createCell(columnCount++);
```

```
163     cell.setCellValue(r.getIntNetDensityStart());
164
165     cell = row.createCell(columnCount++);
166     cell.setCellValue(r.getIntNetDensityFinish());
167
168     cell = row.createCell(columnCount++);
169     cell.setCellValue(r.getDiversityDemandsGenerated());
170
171     cell = row.createCell(columnCount++);
172     cell.setCellValue(r.getDiversityDemandsComplete());
173
174     cell = row.createCell(columnCount++);
175     cell.setCellValue(r.getAgentAbsences());
176
177     cell = row.createCell(columnCount++);
178     cell.setCellValue(r.getNumLeaderInterventions());
179
180     cell = row.createCell(columnCount++);
181     cell.setCellValue(r.getNumLeaderMicroSupports());
182
183     cell = row.createCell(columnCount++);
184     cell.setCellValue(r.getNumAgentsLostLow());
185
186     cell = row.createCell(columnCount++);
187     cell.setCellValue(r.getNumAgentsLostMod());
188
189     cell = row.createCell(columnCount++);
190     cell.setCellValue(r.getNumAgentsLostHigh());
191
192     cell = row.createCell(columnCount++);
193     cell.setCellValue(r.getProjectDuration() / 28800);
194
195    }
196
197  }
198
199 //
200 // // Metric Summaries -- list then data
201 //
202 // private void writeHeaderLineSummaries(XSSFSheet sheet, int runsCount)
        {
203 //
204 //  Row headerRow = sheet.createRow(0);
205 //  headerRow.setRowStyle(styleBold);
206 //
207 //  Cell headerCell = headerRow.createCell(0);
208 //  headerCell.setCellValue("Result ID");
209 //
210 //  headerCell = headerRow.createCell(1);
211 //  headerCell.setCellValue("ObjectID");
212 //
213 //  headerCell = headerRow.createCell(2);
214 //  headerCell.setCellValue("ObjectName");
215 //
```

```
216 //  headerCell = headerRow.createCell(3);
217 //  headerCell.setCellValue("ObjectType");
218 //
219 //  headerCell = headerRow.createCell(4);
220 //  headerCell.setCellValue("Metric");
221 //
222 // }
223 //
224 // private void writeStylesSummaries(XSSFSheet sheet, int runsCount) {
225 //
226 //  sheet.autoSizeColumn(0);
227 //  sheet.autoSizeColumn(1);
228 //  sheet.autoSizeColumn(2);
229 //  sheet.autoSizeColumn(3);
230 //  sheet.autoSizeColumn(4);
231 //
232 //
233 //  Row row = sheet.getRow(0);
234 //  for(int j = 0; j<row.getLastCellNum(); j++) {
235 //   row.getCell(j).setCellStyle(style);
236 //  }
237 // }
238 //
239 // private void writeRowsSummaries(List<SimResult> results2,
      XSSFWorkbook workbook,XSSFSheet sheet) {
240 //  int rowCount = 1;
241 //
242 //  for (SimResult r : results2) {
243 //   for (int n=0; n < r.getMetricsCount(); n++) {
244 //    Metric m = r.getMetric(n);
245 //
246 //     Row row = sheet.createRow(rowCount++);
247 //
248 //     int columnCount = 0;
249 //
250 //     Cell cell = row.createCell(columnCount++);
251 //     cell.setCellValue(r.getShortID());
252 //
253 //     cell = row.createCell(columnCount++);
254 //     cell.setCellValue(m.getModelObjectID().toString());
255 //
256 //     cell = row.createCell(columnCount++);
257 //     cell.setCellValue(m.getModelObjectName());
258 //
259 //     cell = row.createCell(columnCount++);
260 //     cell.setCellValue(m.getModelObjectTypeSimpleName());
261 //
262 //     cell = row.createCell(columnCount++);
263 //     cell.setCellValue(m.getName());
264 //
265 //    }
266 //   }
267 //
268 //  }
```

```
269 //
270 // private void writeHeaderLineSummariesData(XSSFSheet sheet, int
        runsCount) {
271 //
272 //  Row headerRow = sheet.createRow(0);
273 //  headerRow.setRowStyle(styleBold);
274 //
275 //  Cell headerCell = headerRow.createCell(0);
276 //  headerCell.setCellValue("Result ID");
277 //
278 //  headerCell = headerRow.createCell(1);
279 //  headerCell.setCellValue("ObjectID");
280 //
281 //  headerCell = headerRow.createCell(2);
282 //  headerCell.setCellValue("ObjectName");
283 //
284 //  headerCell = headerRow.createCell(3);
285 //  headerCell.setCellValue("ObjectType");
286 //
287 //  headerCell = headerRow.createCell(4);
288 //  headerCell.setCellValue("Metric");
289 //
290 //  headerCell = headerRow.createCell(5);
291 //  headerCell.setCellValue("Type");
292 //
293 //  headerCell = headerRow.createCell(6);
294 //  headerCell.setCellValue("Runs");
295 //
296 //  for(int n= 0; n<runsCount; n++ ) {
297 //
298 //   headerCell = headerRow.createCell(7+n);
299 //   headerCell.setCellValue(n+1);
300 //  }
301 //
302 //
303 // }
304 //
305 // private void writeStylesSummariesData(XSSFSheet sheet, int runsCount)
        {
306 //
307 //  sheet.autoSizeColumn(0);
308 //  sheet.autoSizeColumn(1);
309 //  sheet.autoSizeColumn(2);
310 //  sheet.autoSizeColumn(3);
311 //  sheet.autoSizeColumn(4);
312 //  sheet.autoSizeColumn(5);
313 //  sheet.autoSizeColumn(6);
314 //
315 //  for(int n= 0; n<runsCount; n++ ) {
316 //   sheet.autoSizeColumn(7+n);
317 //  }
318 //
319 //
320 //  Row row = sheet.getRow(0);
```

```
321 //  for(int j = 0; j<row.getLastCellNum(); j++) {
322 //   row.getCell(j).setCellStyle(style);
323 //  }
324 // }
325 //
326 // private void writeRowsSummariesData(List<SimResult> results2,
        XSSFWorkbook workbook,XSSFSheet sheet) {
327 //  int rowCount = 1;
328 //
329 //  for (SimResult r : results2) {
330 //   for (int n=0; n < r.getMetricsCount(); n++) {
331 //    Metric m = r.getMetric(n);
332 //
333 //    for (int stat=0; stat < m.getSummariesCount(); stat++) {
334 //     MetricSummary ms = m.getSummary(stat);
335 //
336 //
337 //     Row row = sheet.createRow(rowCount++);
338 //
339 //     int columnCount = 0;
340 //
341 //     Cell cell = row.createCell(columnCount++);
342 //     cell.setCellValue(r.getShortID());
343 //
344 //     cell = row.createCell(columnCount++);
345 //     cell.setCellValue(m.getModelObjectID().toString());
346 //
347 //     cell = row.createCell(columnCount++);
348 //     cell.setCellValue(m.getModelObjectName());
349 //
350 //     cell = row.createCell(columnCount++);
351 //     cell.setCellValue(m.getModelObjectTypeSimpleName());
352 //
353 //     cell = row.createCell(columnCount++);
354 //     cell.setCellValue(m.getName());
355 //
356 //     cell = row.createCell(columnCount++);
357 //     cell.setCellValue(ms.getType().name());
358 //
359 //     cell = row.createCell(columnCount++);
360 //     cell.setCellValue(ms.getRunsCount());
361 //
362 //     for(int run=0; run < ms.getRunsCount(); run++) {
363 //      cell = row.createCell(columnCount++);
364 //      cell.setCellValue(ms.getValue(run));
365 //
366 //     }
367 //    }
368 //   }
369 //
370 //  }
371 // }
372 //
373 //
```

```
374 // // Metric TIme Series -- list then data
375 //
376 //
377 // private void writeHeaderSeries(XSSFSheet sheet) {
378 //
379 //  Row headerRow = sheet.createRow(0);
380 //  headerRow.setRowStyle(styleBold);
381 //
382 //  Cell headerCell = headerRow.createCell(0);
383 //  headerCell.setCellValue("Result ID");
384 //
385 //  headerCell = headerRow.createCell(1);
386 //  headerCell.setCellValue("ObjectID");
387 //
388 //  headerCell = headerRow.createCell(2);
389 //  headerCell.setCellValue("ObjectName");
390 //
391 //  headerCell = headerRow.createCell(3);
392 //  headerCell.setCellValue("ObjectType");
393 //
394 //  headerCell = headerRow.createCell(4);
395 //  headerCell.setCellValue("Metric");
396 //
397 //  headerCell = headerRow.createCell(5);
398 //  headerCell.setCellValue("EmptysAsZero?");
399 //
400 // }
401 //
402 // private void writeStylesSeries(XSSFSheet sheet) {
403 //
404 //  sheet.autoSizeColumn(0);
405 //  sheet.autoSizeColumn(1);
406 //  sheet.autoSizeColumn(2);
407 //  sheet.autoSizeColumn(3);
408 //  sheet.autoSizeColumn(4);
409 //  sheet.autoSizeColumn(5);
410 //
411 //  Row row = sheet.getRow(0);
412 //  for(int j = 0; j<row.getLastCellNum(); j++) {
413 //   row.getCell(j).setCellStyle(style);
414 //  }
415 // }
416 //
417 // private void writeRowsSeries(List<SimResult> results, XSSFWorkbook
      workbook,XSSFSheet sheet) {
418 //  int rowCount = 1;
419 //
420 //  for (SimResult r : results) {
421 //   for (int n=0; n < r.getMetricsCount(); n++) {
422 //    Metric m = r.getMetric(n);
423 //     MetricTimeSeries ts = m.getSeries();
424 //     if (ts == null) {
425 //      continue;
426 //     }
```

```
427 //
428 //      Row row = sheet.createRow(rowCount++);
429 //
430 //      int columnCount = 0;
431 //
432 //      Cell cell = row.createCell(columnCount++);
433 //      cell.setCellValue(r.getShortID());
434 //
435 //      cell = row.createCell(columnCount++);
436 //      cell.setCellValue(m.getModelObjectID().toString());
437 //
438 //      cell = row.createCell(columnCount++);
439 //      cell.setCellValue(m.getModelObjectName());
440 //
441 //      cell = row.createCell(columnCount++);
442 //      cell.setCellValue(m.getModelObjectTypeSimpleName());
443 //
444 //      cell = row.createCell(columnCount++);
445 //      cell.setCellValue(m.getName());
446 //
447 //      cell = row.createCell(columnCount++);
448 //      cell.setCellValue(ts.isEmptyAsZero());
449 //
450 //    }
451 //
452 //   }
453 // }
454 //
455 // // Series Data sheet
456 //
457 // private void writeHeaderSeriesData(XSSFSheet sheet) {
458 //
459 //  Row headerRow = sheet.createRow(0);
460 //  headerRow.setRowStyle(styleBold);
461 //
462 //  Cell headerCell = headerRow.createCell(0);
463 //  headerCell.setCellValue("Result ID");
464 //
465 //  headerCell = headerRow.createCell(1);
466 //  headerCell.setCellValue("ObjectID");
467 //
468 //  headerCell = headerRow.createCell(2);
469 //  headerCell.setCellValue("ObjectName");
470 //
471 //  headerCell = headerRow.createCell(3);
472 //  headerCell.setCellValue("ObjectType");
473 //
474 //  headerCell = headerRow.createCell(4);
475 //  headerCell.setCellValue("Metric");
476 //
477 //  headerCell = headerRow.createCell(5);
478 //  headerCell.setCellValue("Time");
479 //
480 //  headerCell = headerRow.createCell(6);
```

```
481 //  headerCell.setCellValue("Min");
482 //
483 //  headerCell = headerRow.createCell(7);
484 //  headerCell.setCellValue("Avg");
485 //
486 //  headerCell = headerRow.createCell(8);
487 //  headerCell.setCellValue("Max");
488 //
489 //  headerCell = headerRow.createCell(9);
490 //  headerCell.setCellValue("StDev");
491 //
492 //
493 // }
494 //
495 // private void writeStylesSeriesData(XSSFSheet sheet) {
496 //
497 //  sheet.autoSizeColumn(0);
498 //  sheet.autoSizeColumn(1);
499 //  sheet.autoSizeColumn(2);
500 //  sheet.autoSizeColumn(3);
501 //  sheet.autoSizeColumn(4);
502 //  sheet.autoSizeColumn(5);
503 //  sheet.autoSizeColumn(6);
504 //  sheet.autoSizeColumn(7);
505 //  sheet.autoSizeColumn(8);
506 //  sheet.autoSizeColumn(9);
507 //
508 //
509 //  Row row = sheet.getRow(0);
510 //  for(int j = 0; j<row.getLastCellNum(); j++) {
511 //   row.getCell(j).setCellStyle(style);
512 //  }
513 // }
514 //
515 // private void writeRowsSeriesData(List<SimResult> results,
    XSSFWorkbook workbook,XSSFSheet sheet) {
516 //  int rowCount = 1;
517 //
518 //  for (SimResult r : results) {
519 //   for (int n=0; n < r.getMetricsCount(); n++) {
520 //    Metric m = r.getMetric(n);
521 //     MetricTimeSeries ts = m.getSeries();
522 //     if (ts == null) {
523 //      continue;
524 //     }
525 //     for(long t=ts.getStart(); t <= ts.getEnd(); t += ts.getPeriod())
    {
526 //
527 //      Row row = sheet.createRow(rowCount++);
528 //
529 //      int columnCount = 0;
530 //
531 //      Cell cell = row.createCell(columnCount++);
532 //      cell.setCellValue(r.getShortID());
```

```
533 //
534 //      cell = row.createCell(columnCount++);
535 //      cell.setCellValue(m.getModelObjectID().toString());
536 //
537 //      cell = row.createCell(columnCount++);
538 //      cell.setCellValue(m.getModelObjectName());
539 //
540 //      cell = row.createCell(columnCount++);
541 //      cell.setCellValue(m.getModelObjectTypeSimpleName());
542 //
543 //      cell = row.createCell(columnCount++);
544 //      cell.setCellValue(m.getName());
545 //
546 //      cell = row.createCell(columnCount++);
547 //      cell.setCellValue(t);
548 //
549 //      cell = row.createCell(columnCount++);
550 //      cell.setCellValue(ts.getMin(t));
551 //
552 //      cell = row.createCell(columnCount++);
553 //      cell.setCellValue(ts.getAvg(t));
554 //
555 //      cell = row.createCell(columnCount++);
556 //      cell.setCellValue(ts.getMax(t));
557 //
558 //      cell = row.createCell(columnCount++);
559 //      cell.setCellValue(ts.getStdDev(t));
560 //
561 //     }
562 //    }
563 //
564 //   }
565 // }
566 //
567 // // Decisions sheet
568 //
569 // private void writeHeaderDecisions(XSSFSheet sheet) {
570 //
571 //  Row headerRow = sheet.createRow(0);
572 //  headerRow.setRowStyle(styleBold);
573 //
574 //  Cell headerCell = headerRow.createCell(0);
575 //  headerCell.setCellValue("Result ID");
576 //
577 //  headerCell = headerRow.createCell(1);
578 //  headerCell.setCellValue("Result Name");
579 //
580 //  headerCell = headerRow.createCell(2);
581 //  headerCell.setCellValue("Type");
582 //
583 //  headerCell = headerRow.createCell(3);
584 //  headerCell.setCellValue("Name");
585 //
586 //  headerCell = headerRow.createCell(4);
```

```
587 //  headerCell.setCellValue("Value");
588 //
589 //  // key
590 //
591 //  headerCell = headerRow.createCell(5);
592 //  headerCell = headerRow.createCell(6);
593 //  headerCell.setCellValue("Key:");
594 //  headerCell = headerRow.createCell(7);
595 //  headerCell.setCellValue("Populations");
596 //
597 // }
598 //
599 // private void writeStylesDecisions(XSSFSheet sheet) {
600 //
601 //  sheet.autoSizeColumn(0);
602 //  sheet.autoSizeColumn(1);
603 //  sheet.autoSizeColumn(2);
604 //  sheet.autoSizeColumn(3);
605 //  sheet.autoSizeColumn(4);
606 //  sheet.autoSizeColumn(5);
607 //  sheet.autoSizeColumn(6);
608 //  sheet.autoSizeColumn(7);
609 //
610 //  Row row = sheet.getRow(0);
611 //  for(int j = 0; j<row.getLastCellNum(); j++) {
612 //   if (j < 5) {
613 //    row.getCell(j).setCellStyle(style);
614 //   }
615 //   if (j==5) { // this is the break before the key
616 //    continue;
617 //   }
618 //   if (j > 5) {
619 //    row.getCell(j).setCellStyle(keyStyle);
620 //   }
621 //  }
622 // }
623 //
624 // private String getResidenceString(Boolean[] res) {
625 //  String retV = "";
626 //  for (Boolean b : res) {
627 //   retV += b.toString() + " ";
628 //  }
629 //  return retV;
630 // }
631 //
632 // private void writeRowsDecisions(List<SimResult> results, XSSFWorkbook
      workbook,XSSFSheet sheet) {
633 //  int rowCount = 1;
634 //
635 //  for (SimResult r : results) {
636 //   DecisionSet ds = r.getDecisions();
637 //   if (ds == null) {
638 //    return;
639 //   }
```

```
640 //
641 //    var res = ds.getResidenceDecisions();
642 //    if (res == null) {
643 //     return;
644 //    }
645 //
646 //    for (Entry<String, Boolean[]> resE : res.entrySet()) {
647 //       Row row = sheet.createRow(rowCount++);
648 //
649 //       int columnCount = 0;
650 //
651 //       Cell cell = row.createCell(columnCount++);
652 //       cell.setCellValue(r.getShortID());
653 //
654 //       cell = row.createCell(columnCount++);
655 //       cell.setCellValue(r.getName());
656 //
657 //       cell = row.createCell(columnCount++);
658 //       cell.setCellValue("residence");
659 //
660 //       cell = row.createCell(columnCount++);
661 //       cell.setCellValue(resE.getKey());
662 //
663 //       cell = row.createCell(columnCount++);
664 //       cell.setCellValue(getResidenceString(resE.getValue()));
665 //
666 //       // key
667 //       if (rowCount == 2) {
668 //        cell = row.createCell(columnCount++);
669 //        cell = row.createCell(columnCount++);
670 //        for (String p : ds.getPopulationNames()) {
671 //         cell = row.createCell(columnCount++);
672 //         cell.setCellValue(p);
673 //         cell.setCellStyle(keyStyle);
674 //
675 //        }
676 //       }
677 //    }
678 //
679 //    var spaces = ds.getSpaceDecisions();
680 //    for (Entry<String, ActivityType> spaceE : spaces.entrySet()) {
681 //       Row row = sheet.createRow(rowCount++);
682 //
683 //       int columnCount = 0;
684 //
685 //       Cell cell = row.createCell(columnCount++);
686 //       cell.setCellValue(r.getShortID());
687 //
688 //       cell = row.createCell(columnCount++);
689 //       cell.setCellValue(r.getName());
690 //
691 //       cell = row.createCell(columnCount++);
692 //       cell.setCellValue("spaces");
693 //
```

```
694 //       cell = row.createCell(columnCount++);
695 //       cell.setCellValue(spaceE.getKey());
696 //
697 //       cell = row.createCell(columnCount++);
698 //       cell.setCellValue(spaceE.getValue().toString());
699 //    }
700 //    var modes = ds.getModeDecisions();
701 //    for (Entry<String, TransitMode> modesE : modes.entrySet()) {
702 //       Row row = sheet.createRow(rowCount++);
703 //
704 //       int columnCount = 0;
705 //
706 //       Cell cell = row.createCell(columnCount++);
707 //       cell.setCellValue(r.getShortID());
708 //
709 //       cell = row.createCell(columnCount++);
710 //       cell.setCellValue(r.getName());
711 //
712 //       cell = row.createCell(columnCount++);
713 //       cell.setCellValue("modes");
714 //
715 //       cell = row.createCell(columnCount++);
716 //       cell.setCellValue(modesE.getKey());
717 //
718 //       cell = row.createCell(columnCount++);
719 //       cell.setCellValue(modesE.getValue().toString());
720 //    }
721 //   }
722 // }
723 //
724
725 }
```

A.3 Agent Files

A.3.1 Agent.java

```
1
2  import java.util.ArrayList;
3  import java.util.Comparator;
4  import java.util.HashMap;
5  import java.util.List;
6  import java.util.Map;
7  import java.util.Random;
8  import java.util.UUID;
9
10 import javafx.beans.property.BooleanProperty;
11 import javafx.beans.property.DoubleProperty;
12 import javafx.beans.property.IntegerProperty;
13 import javafx.beans.property.ListProperty;
14 import javafx.beans.property.ObjectProperty;
15 import javafx.beans.property.SimpleBooleanProperty;
16 import javafx.beans.property.SimpleDoubleProperty;
```

```
17 import javafx.beans.property.SimpleIntegerProperty;
18 import javafx.beans.property.SimpleListProperty;
19 import javafx.beans.property.SimpleObjectProperty;
20 import javafx.beans.property.SimpleStringProperty;
21 import javafx.beans.property.StringProperty;
22 import javafx.collections.FXCollections;
23 import javafx.collections.ObservableList;
24
25 /**
26  *
27  * @author Jeremy Goodwin with considerable support from Dr. Bryan Moser
       Written
28  *         in conjunction with thesis pursuant to MIT SDM degree, 2021
29  *
30  *         This class controls establishes supplies and controls an
       agent for a
31  *         diversity-related agent-based model. It establishes the rules
32  *         associated with when agents will generate diversity-related
       concerns,
33  *         when they will take the day off, and when they will no longer
        serve
34  *         on the project team. Additionally, it also establishes the
       rules
35  *         associated with transformational leader interventions and
       daily
36  *         support actions.
37  *
38  */
39 public class Agent {
40
41  // Agent Parameters
42  UUID id;
43  StringProperty name;
44  ObjectProperty<AgentType> type;
45  ObjectProperty<AgentMode> mode; // the mode of the agent in the
       attention cycle
46
47  // Leadership Parameters
48  BooleanProperty transformationalLeader;
49  BooleanProperty sociometricBadgeAwareness;
50  IntegerProperty numInterventions;
51  IntegerProperty numMicroSupports;
52
53  // Identity Diversity Parameters
54  ObjectProperty<Race> race;
55  ObjectProperty<Gender> gender;
56  BooleanProperty lowRepresentation;
57  BooleanProperty moderateRepresentation;
58  ObjectProperty<DiversityApproach> diversityApproach;
59
60  // Interaction Tracking Tables
61  HashMap<Agent, Boolean> jointInterventionTable = new HashMap<Agent,
       Boolean>();
62
```

```
63  // Demand Lists
64  ListProperty<DemandFX> internalAssignedDemands; // agent to-do list
65  ListProperty<DemandFX> internalDemandsPending; // internal set of
      demands for selection towards completion
66  ListProperty<DemandFX> internalDemandsActive; // internal set of
      demands selected and not yet complete
67  ListProperty<DemandFX> internalDemandsMoot; // internal set of demands
      no longer eligible (complete or past)
68
69  // Collaboration signals
70  ListProperty<DemandFX> internalEmergingSignals; // calls for
      collaboration
71
72  // Network density and diversity consequences
73  HashMap<Agent, Double> agentIntDensityTable = new HashMap<Agent, Double
      >();
74  double aveIntDensity;
75  int postInterventionPeriod;
76
77  ListProperty<Double> aveIntDensityList;
78  ListProperty<DemandFX> diversityConsequences; // collect diversity
      concerns for UI viewing
79  DoubleProperty consequenceThreshold;
80  IntegerProperty agentAbsences; // count the number of times an agent
      stays home
81  DoubleProperty absenceThreshold; // established in session class with
      project length
82
83  // Agent Resources
84  ListProperty<SupplyFX> supplies; // this agent's abilities
85
86  // Time variables
87  IntegerProperty agentTime; // current agent time stamp
88  IntegerProperty agentHorizon; // as far as the agent can see
89  IntegerProperty agentTimeActing; // total time agent is acting
90  IntegerProperty timeStep; // timeStep set by User Interface
91  IntegerProperty agentLoopDelay; // a pause added to the agent loop for
      simulation observation
92
93  // Action variables
94  ObjectProperty<DemandFX> selectedDemand;
95  ObjectProperty<SupplyFX> selectedSupply;
96  ObjectProperty<ActFX> inProcessAct;
97  ListProperty<ActFX> acts; // simply for archive, or possible learning,
      of past acts
98
99  // DEFAULT CONSTRUCTOR (for testing purposes)
100 public Agent(String _name, Race _race, Gender _gender) {
101
102  this(UUID.randomUUID(), new SimpleStringProperty(_name),
103    new SimpleObjectProperty<AgentType>(AgentType.ENGINEER), new
      SimpleObjectProperty<Race>(_race),
104    new SimpleObjectProperty<Gender>(_gender));
105  // new SimpleObjectProperty<EngagementState>(EngagementState.HIGH));
```

```
106 }
107
108 // STANDARD CONSTRUCTOR
109 public Agent(StringProperty _name, ObjectProperty<AgentType> _type) {
110  // ObjectProperty<EngagementState> _engagementState) {
111  this(UUID.randomUUID(), _name, _type, new SimpleObjectProperty<Race>(
      Race.getRandomRace()),
112    new SimpleObjectProperty<Gender>(Gender.getRandomGender()));// ,
      _engagementState);
113 }
114
115 // FULL CONSTRUCTOR
116 public Agent(UUID id, StringProperty _name, ObjectProperty<AgentType>
      _type, ObjectProperty<Race> _race,
117   ObjectProperty<Gender> _gender) {// , ObjectProperty<EngagementState>
       _engagementState) {
118
119  // Agent States
120  id = UUID.randomUUID();
121  name = _name;
122  type = _type;
123  race = _race;
124  gender = _gender;
125  setLowRepresentation();
126  setModerateRepresentation();
127
128  mode = new SimpleObjectProperty<AgentMode>(AgentMode.INACTIVE);
129
130  // Establish supplies
131  supplies = new SimpleListProperty<SupplyFX>(FXCollections.
      observableArrayList());
132  setInitialSupplies(type);
133
134  // Establish internal demands & signals
135  internalAssignedDemands = new SimpleListProperty<DemandFX>(
      FXCollections.observableArrayList());
136  internalEmergingSignals = new SimpleListProperty<DemandFX>(
      FXCollections.observableArrayList());
137  internalDemandsPending = new SimpleListProperty<DemandFX>(
      FXCollections.observableArrayList());
138  internalDemandsActive = new SimpleListProperty<DemandFX>(FXCollections
      .observableArrayList());
139  internalDemandsMoot = new SimpleListProperty<DemandFX>(FXCollections.
      observableArrayList());
140
141  // Establish initial diversity parameters
142  diversityApproach = new SimpleObjectProperty<DiversityApproach>(
      DiversityApproach.EQUALITY);
143  diversityConsequences = new SimpleListProperty<DemandFX>(FXCollections
      .observableArrayList());
144  consequenceThreshold = new SimpleDoubleProperty(0);
145  agentAbsences = new SimpleIntegerProperty(0);
146  absenceThreshold = new SimpleDoubleProperty(0);
147
```

```
148   aveIntDensityList = new SimpleListProperty<Double>(FXCollections.
        observableArrayList());
149
150   // Time states
151   agentTime = new SimpleIntegerProperty(0);
152   agentTimeActing = new SimpleIntegerProperty(0);
153   agentHorizon = new SimpleIntegerProperty(3600 * 8); // 1 work day
154   agentLoopDelay = new SimpleIntegerProperty(0);
155
156   // Action States
157   acts = new SimpleListProperty<ActFX>(FXCollections.observableArrayList
        ());
158   selectedDemand = new SimpleObjectProperty<DemandFX>();
159   selectedSupply = new SimpleObjectProperty<SupplyFX>();
160   inProcessAct = new SimpleObjectProperty<ActFX>();
161
162   // Leader-specific
163   numInterventions = new SimpleIntegerProperty(0);
164   numMicroSupports = new SimpleIntegerProperty(0);
165  }
166
167  // SIMULATION EXECUTION CONTROL METHODS
168  public void start() {
169
170   // setting agent initial time;
171   agentTime.set(0);
172
173   // get to work
174   loop();
175
176   // after each loop, this method simulates generation
177   // of a diversity concern. It simulates team members getting
178   // a chance to say if they had any concerns 1/day
179   this.generateRandomDiversityConcern();
180  }
181
182  public void continueSim() {
183
184   // reset supplies for this time period's work
185   this.supplies.clear();
186   this.setInitialSupplies(this.typeProperty());
187
188   // back to work, except for departed agents
189   if (this.mode.get() != AgentMode.DEPARTED) {
190    loop();
191   }
192   // see comment in start() method
193   this.generateRandomDiversityConcern();
194  }
195
196  // AGENT CONTROL LOOP
197  // This loop governs how the agent will operate throughout the time
       step.
198  // Based on the agent's mode, it directs the agent to a specific method
```

```
    .
199 public void loop() {
200
201
202  int startTime = 0;
203  int nextPause = agentTime.get() + timeStep.get();
204
205  while (agentTime.get() <= nextPause) {
206   switch (mode.get()) {
207   case INACTIVE:
208    mode.set(AgentMode.OBSERVING);
209    break;
210   case WAITING:
211    mode.set(Wait());
212    break;
213   case OBSERVING: // observe environment, load tasks
214    mode.set(Observe((this.agentHorizon.get() - startTime)));
215    break;
216   case SELECTING: // select task to work
217    mode.set(Select());
218    break;
219   case ACTING: // act on the task
220    mode.set(Act());
221    break;
222   case FINISHING: // wrap up the task
223    mode.set(Finish());
224    break;
225   case RECOVERING: // take a day off
226    mode.set(StayHome());
227    break;
228   case SUPPORTING: // leader: provide support
229    mode.set(Support());
230    break;
231   case DEPARTED: // quit
232    mode.set(Depart());
233    break;
234   case COMPLETING: // complete the time step
235    Complete();
236    return;
237   case PAUSED: // pause and exit the loop
238
239    return;
240   default:
241   }
242
243   if (nextPause <= agentTime.get() && timeStep.get() != 0 && mode.get()
       != AgentMode.DEPARTED) {
244    mode.set(AgentMode.PAUSED);
245   }
246
247   try { // pause so that the steps can be seen in the UI
248     // System.out.println("Sleeping: " + agentLoopDelay.get() * 100);
249     // This doesn't work yet...
250    Thread.sleep(agentLoopDelay.get() * 0);
```

```
251     } catch (InterruptedException ex) {
252     }
253
254    }
255
256    return;
257   }
258
259   // METHODS BY AGENT MODE
260   private AgentMode Wait() {
261    agentTime.set(agentTime.get() + 5 * 60); // 5 minutes to wait, then
        back to observe
262    return AgentMode.OBSERVING;
263   }
264
265   private AgentMode Observe(int timeRemaining) {
266
267
268    // If leader, check if team needs my support (transformational leader
        behavior)
269    if (this.getType() == AgentType.LEADER && this.transformationalLeader.
        get()) {
270
271     if (this.teamNeedsSupport()) {
272
273      Random rand = new Random();
274      // Choose whether or not to provide support
275      if (rand.nextInt(100) < 75) {
276
277       return AgentMode.SUPPORTING;
278      }
279     }
280    }
281
282    // As an Engineer Agent, should I stay home today?
283    // These decisions are driven by the two methods below
284    // Please see them for more info
285    if (this.diversityConsequences.size() > 0) {
286     if (this.generateAgentDeparture()) {
287      return AgentMode.DEPARTED;
288     } else {
289      // ... or maybe I just need the day off?
290      if (this.generateRandomDayOff()) {
291       return AgentMode.RECOVERING;
292      }
293     }
294    }
295
296    // Agent decides to stay - Let's load up the task list. High-pri tasks
        first
297    for (DemandFX d : internalAssignedDemands) {
298     if (d.getPriority() == 1) {
299      if (!internalDemandsPending.contains(d) && d.getState() !=
        DemandState.COMPLETE) {
```

```
300       if (d.getEffort() <= timeRemaining) {
301
302        // Check to see if supplies match demand
303        for (SupplyFX supply : supplies) {
304         if (supply.isMatch(d)) {
305          internalDemandsPending.add(d);
306          d.setState(DemandState.PENDING);
307          timeRemaining -= d.getEffort();
308         }
309        }
310
311       } else {
312        // Stop loading up tasks once agent to-do list extends beyond time
      step
313         agentTime.set(agentTime.get() + 300);
314         return AgentMode.SELECTING;
315       }
316      }
317     }
318    }
319
320    // Now let's look at remaining tasks
321    for (DemandFX d : internalAssignedDemands) {
322     if (!internalDemandsPending.contains(d) && d.getState() !=
      DemandState.COMPLETE) {
323      if (d.getEffort() <= timeRemaining) {
324       for (SupplyFX supply : supplies) {
325        if (supply.isMatch(d)) {
326         internalDemandsPending.add(d);
327         d.setState(DemandState.PENDING);
328         timeRemaining -= d.getEffort();
329        }
330       }
331      } else {
332       // Stop loading up tasks once agent to-do list extends beyond time
      step
333        agentTime.set(agentTime.get() + 300);
334        return AgentMode.SELECTING;
335      }
336     }
337
338    }
339    agentTime.set(agentTime.get() + 300);
340    return AgentMode.SELECTING;
341   }
342
343   private AgentMode Select() {
344
345    if (internalDemandsPending.size() > 0) {
346
347     internalDemandsPending.stream().sorted(new DemandPriorityComparator()
      );
348
349     for (DemandFX d : internalDemandsPending) { // we will here just find
```

```
      the first match
350
351     for (SupplyFX s : supplies) {
352      if (s.isMatch(d)) {
353       selectedDemand.set(d); // we found a matching supply
354       selectedSupply.set(s);
355       internalDemandsPending.remove(d);
356       internalDemandsActive.add(d);
357       d.setState(DemandState.ACTIVE);
358       d.setRequestor(new SimpleObjectProperty<Agent>(this));
359
360       return AgentMode.ACTING;
361      }
362     }
363    }
364   }
365
366   return AgentMode.WAITING;
367  }
368
369  private AgentMode Act() {
370   ActFX a = new ActFX();
371
372   a.GetSet(agentTime.get(), this, selectedSupply.get(), selectedDemand.
       get());
373   inProcessAct.set(a);
374
375   selectedSupply.get().attempt(a);
376
377   if (selectedDemand.get().getType() == ActType.COORD) {
378   }
379
380   if (selectedDemand.get().getType() == ActType.COORD) {
381    Agent helperAgent = this.callForHelp(selectedDemand.get());
382    a.GetSet(agentTime.get(), helperAgent, selectedSupply.get(),
       selectedDemand.get());
383
384   }
385
386   return AgentMode.FINISHING;
387
388  }
389
390  private AgentMode Finish() {
391   // What effort was required? How much time has passed?
392   agentTime.set(inProcessAct.get()._timeStop);
393   agentTimeActing.set(agentTimeActing.get() + inProcessAct.get()._effort
       );
394   acts.add(inProcessAct.get());
395
396   internalDemandsPending.remove(selectedDemand.get());
397   internalDemandsActive.remove(selectedDemand.get());
398   internalDemandsMoot.add(selectedDemand.get());
399
```

```
400   selectedDemand.get().setState(DemandState.COMPLETE);
401
402   selectedDemand.set(null);
403   selectedSupply.set(null);
404   inProcessAct.set(null);
405
406   // TODO - partial completion of demand
407   return AgentMode.SELECTING;
408  }
409
410  private AgentMode StayHome() {
411   // This method prevents the agent from working on any tasks during the
        time
412   // period
413   this.setAgentAbsences(getAgentAbsences() + 1);
414
415   // test print
416 //  System.out.println(
417 //    this.getName() + ": Day off (" + this.getAgentAbsences() + ")
      ................................");
418
419   agentTime.set(agentTime.get() + this.timeStep.get()); // period off
420   return AgentMode.PAUSED;
421  }
422
423  private AgentMode Depart() {
424   // This method sets the agent mode to DEPARTED so that it can be
       collected
425   // in the environment (i.e., World) and prevented from further work on
        this
426   // project
427   agentTime.set(agentTime.get() + this.timeStep.get());
428
429   // test print
430 //  System.out.println(this.getName() + " .......... I quit!!
      ..........");
431
432   return AgentMode.DEPARTED;
433  }
434
435  private AgentMode Support() {
436   // This method can only be executed by Leader agents
437   // TODO: Change if LEAD supplies are added to engineer agents
438
439   Random rand = new Random();
440
441   // Each agent has an internal network density (i.e., tie strength)
       with other
442   // agents
443   // in its agentIntDensityTable. As the agent knows its tie strength
       with the
444   // other
445   // agents, this is a convenient way for agents to know what to do with
        respect
```

```
446   // to how
447   // they're feeling towards the other agents
448   List<Agent> keysAsArray = new ArrayList<Agent>(this.
       agentIntDensityTable.keySet());
449   keysAsArray.remove(this);
450
451   // For leaders without sociometric badge awareness...
452   if (!this.sociometricBadgeAwareness.get()) {
453    // Leader does his/her own work until 1/2 absence threshold is
       triggered.
454    // Diversity Consequences size is also an important part of this
       condition
455    // because
456    // they will be mitigated for agents that pass the 50% threshold. If
       the
457    // diversity
458    // consequences variable was taken out of this conditional, the
       leader agent
459    // would
460    // never stop conducting interventions with agents past the 50%
       threshold.
461    for (Agent agent : keysAsArray) {
462     if (agent.getAgentAbsences() > 0.5 * agent.getAbsenceThreshold()
463       && agent.diversityConsequences.size() > 0.5 * agent.
       getConsequenceThreshold()) {
464
465      this.generateLeaderIntevention(agent);
466
467     }
468    }
469
470    // Post intervention: - use available leadership capacity per day to
       "support"
471    // team
472    if (!this.jointInterventionTable.isEmpty()) {
473     // brute force method of extracting leadership capacity
474     for (SupplyFX supply : this.supplies) {
475      if (supply.getType() == ActType.LEAD) {
476       if (supply.getCapacity() > 0) {
477
478        Agent chosenAgent = this; // initialize chosenAgent
479
480        // Priority 1: Help those that are most upset
481        // Use awareness of absences to drive intervention
482        for (Agent agent : keysAsArray) {
483         if (agent.getAgentAbsences() > chosenAgent.getAgentAbsences()
484           && agent.diversityConsequences.size() > 0) {
485          chosenAgent = agent;
486         } else {
487
488         }
489        }
490        if (chosenAgent != this) {
491
```

```
492         // conduct daily support and subtract from day's support
    capacity
493         supply.setCapacity(supply.getCapacity() - this.
    generateMicroSupport(chosenAgent));
494       }
495
496       // Priority 2: use knowledge from intervention to address highest
     concerned
497       // agents
498       // Note: this currently needs to be the same as the conditional
    in
499       // teamStruggling() method
500       chosenAgent = this;
501
502       for (Agent agent : keysAsArray) {
503         if (agent.diversityConsequences.size() > chosenAgent.
    diversityConsequences.size()) {
504           chosenAgent = agent;
505         }
506       }
507       if (chosenAgent != this) {
508         supply.setCapacity(supply.getCapacity() - this.
    generateMicroSupport(chosenAgent));
509
510
511       }
512
513       // Priority 3: Check to see if random agent needs support
514       // Note: hard-coded cap at 3 to brute force limit to leader agent
    's
515       // LEAD supply
516       for (int i = 1; i < 3; i++) {
517         Agent randomAgent = chooseAgent();
518
519         if (!randomAgent.diversityConsequences.isEmpty()) {
520           supply.setCapacity(supply.getCapacity() - this.
    generateMicroSupport(randomAgent));
521
522         }
523       }
524     }
525   }
526  }
527 }
528
529 } else {
530  // For leaders WITH sociometric badge awareness
531  for (Agent agent : keysAsArray) {
532   if (agent.diversityConsequences.size() > 0) {
533    for (SupplyFX supply : this.supplies) {
534     if (supply.getType() == ActType.LEAD && supply.getCapacity() > 0)
    {
535
536       supply.setCapacity(supply.getCapacity() - this.
```

```
    generateMicroSupport(agent));
537
538         }
539       }
540     }
541   }
542  }
543  return AgentMode.OBSERVING;
544 }
545
546 private AgentMode Complete() {
547  return null;
548 }
549
550 // COORDINATION METHODS
551 private Agent chooseAgent() {
552
553  Random rand = new Random(); // ensures agents don't always pick same
      agent
554
555  List<Agent> keysAsArray = new ArrayList<Agent>(this.
      agentIntDensityTable.keySet());
556  keysAsArray.remove(this);
557
558  ArrayList<Agent> dissimilarAgents = new ArrayList<>();
559
560  for (Agent agent : keysAsArray) {
561
562   // Assumption: working coordination demands should need someone with
      dissimilar
563   // skill set
564   if (this.race.get() != agent.race.get() || this.gender.get() != agent
      .gender.get()) {
565    if (agent.getType() != AgentType.LEADER) {
566     dissimilarAgents.add(agent);
567    }
568   }
569  }
570
571  // But the team might not have any dissimilar agents left...
572  if (dissimilarAgents.size() == 0) {
573   return keysAsArray.get(rand.nextInt(keysAsArray.size()));
574  }
575
576  // ... If it does, though, return a random agent in that set
577  return dissimilarAgents.get(rand.nextInt(dissimilarAgents.size()));
578
579 }
580
581 private Agent callForHelp(DemandFX d) {
582  /*
583   * Simulates interaction between two agents on a single demand.
      Updates internal
584   * network density between the agent pair (i.e., tie strength).
```

```
585     */
586    // Choose agent to work with
587    Agent chosenAgent = chooseAgent();
588
589    // Brute force to make sure engineer agents work with engineers (since
         they all
590    // have the same skill set) rather than leaders. For future work,
         update this to
591    // include any agent with requisite skills.
592    while (chosenAgent.getType() == AgentType.LEADER) {
593     chosenAgent = chooseAgent();
594     // System.out.println("here");
595    }
596
597    // Now, update current internal network density based on random +/-
         interaction
598    double cid = this.agentIntDensityTable.get(chosenAgent);
599
600    double factor = .05; // boost (+/-) due to interaction
601    Random rand = new Random();  // allow for random effects
602
603    // Unaddressed diversity consequences ==> decreased internal network
         density
604    long chosenAgentConcerns = chosenAgent.diversityConsequences.stream()
605      .filter(e -> e.getState() != DemandState.COMPLETE).count();
606    long myUnaddressedConcerns = this.diversityConsequences.stream()
607      .filter(e -> e.getState() != DemandState.COMPLETE).count();
608
609    // Assumption: Similar agents ==> tie strength boost due to similarity
         /homophily
610    if (this.race.get() == chosenAgent.race.get() && this.gender.get() ==
         chosenAgent.gender.get()) {
611     factor = 2 * factor;
612
613    } else if (this.race.get() == chosenAgent.race.get() || this.gender.
         get() == chosenAgent.gender.get()) {
614     // Assumption: Somewhat similar agent ==> tie strength could be
         positive or
615     // negative
616     if (rand.nextBoolean()) {
617      factor = -factor;
618     }
619
620    } else {
621     // Assumption: Dissimilar agent ==> tie strength change biases
         towards negative
622     if (rand.nextInt(100) < 75) {
623      factor = -factor;
624     }
625    }
626
627    // Assumption: Accumulated concerns ==> negative tie strength changes
         get worse
628    if (factor < 0 && (chosenAgentConcerns > 0 || myUnaddressedConcerns >
```

```
      0)) {
629    factor = 2 * factor; // factor is already negative
630   }
631
632   // Assumption: If leader addresses both agent's diversity concerns,
       tie
633   // strength is only increased
634   if (factor < 0 && (chosenAgentConcerns == 0 && myUnaddressedConcerns
       == 0)) {
635    factor = -factor;
636   }
637
638   // Update cid (current internal network density)
639   cid = Math.min((1 + factor) * cid, .95);
640
641   // Update interaction trackers (density & interactions)
642   this.agentIntDensityTable.put(chosenAgent, cid);
643   chosenAgent.agentIntDensityTable.put(this, cid);
644
645   this.refreshAveIntNetDensity();
646   chosenAgent.refreshAveIntNetDensity();
647
648   // Update interaction tracker with positive/negative trend (true/false
       )
649   // TODO: Create new jointInteractionTable instead of Intervention
       table
650   if (factor >= 0) {
651    this.jointInterventionTable.put(chosenAgent, true);
652    chosenAgent.jointInterventionTable.put(this, true);
653   } else {
654    this.jointInterventionTable.put(chosenAgent, false);
655    chosenAgent.jointInterventionTable.put(this, false);
656   }
657   // test print
658 //  System.out.println(this.getName() + " " + this.race.get() + " " +
       this.gender.get());
659 //  System.out.println(chosenAgent.getName() + " " + chosenAgent.race.
       get() + " " + chosenAgent.gender.get());
660
661   return chosenAgent;
662  }
663
664  public void refreshAveIntNetDensity() {
665   double sum = 0;
666   double count = 0;
667
668   for (Map.Entry<Agent, Double> entry : this.agentIntDensityTable.
       entrySet()) {
669    if (entry.getKey().getType() != AgentType.LEADER && entry.getKey() !=
        this) {
670     count += 1; // allows for more than 1 leader to be taken out of
       calculation
671     sum += entry.getValue();
672    }
```

```
673   }
674   this.aveIntDensity = sum / count;
675   this.aveIntDensityList.add(aveIntDensity);
676  }
677
678  private boolean getInternalDensityTrend() {
679
680   if (this.aveIntDensityList.size() < 2) {
681    return true;
682   }
683
684   int lastIndex = this.aveIntDensityList.size() - 1;
685   int penIndex = this.aveIntDensityList.size() - 2;
686
687   double lastValue = this.aveIntDensityList.get(lastIndex);
688   double penValue = this.aveIntDensityList.get(penIndex);
689
690   if (lastValue - penValue > 0) {
691    return true;
692   } else {
693    return false;
694   }
695  }
696
697  // DIVERSITY CONSEQUENCE METHODS
698  public boolean generateRandomDiversityConcern() {
699
700   boolean environmentFactor = false;
701   double genericPercentage = 10;
702   Random rand = new Random();
703
704
705   if (this.type.get() != AgentType.LEADER) {
706
707    // Assumption: strong ties ==> concerns not likely
708    if (this.aveIntDensity > 0.85) {
709     environmentFactor = false;
710     // Assumption: growing ties ==> concerns less likely
711    } else if (this.getInternalDensityTrend()) {
712     if (rand.nextInt(100) > 25) {
713      environmentFactor = false;
714     }
715
716    } else {
717     // include possibility of diversity concerns regardless of approach
718     // such that any agent could have a concern
719     if (rand.nextInt(100) < genericPercentage) {
720      environmentFactor = true;
721     }
722
723     // Determine if environment leads to diversity consequences
724     switch (this.diversityApproach.get()) {
725     case EQUALITY: // Apfelbaum et al ==> moderate representation show
      more concerns
```

```
726       if (this.getRace() == Race.HIGH_REP && this.getGender() == Gender.
      LOW_REP) {
727          genericPercentage = 25;
728          environmentFactor = true;
729         }
730         break;
731
732       case DIFFERENCE: // Apfelbaum et al ==> low representation show more
       concerns
733         if (this.race.get() == Race.LOW_REP) {
734          genericPercentage = 25;
735          environmentFactor = true;
736         }
737         break;
738       }
739       }
740     }
741
742     if (environmentFactor) {
743
744       if (rand.nextInt(100) < genericPercentage) {
745
746         // Signal diversity-related consequence
747         DemandFX diversityConcern = new DemandBase(ActType.LEAD, 3600, 9,
      17, "Diversity Concern");
748         diversityConcern.setPriority(1);
749         diversityConcern.setRequestor(new SimpleObjectProperty<Agent>(this)
      );
750
751         diversityConsequences.add(diversityConcern);
752
753         // test print
754 //        System.out.println(this.getName() + ": Diversity Concern Added ("
755 //          + this.diversityConsequences.size() + ")
      ..............................");
756         return true;
757       }
758     }
759     return false;
760   }
761
762   public boolean generateRandomDayOff() {
763
764     int lastIndex = diversityConsequences.getSize() - 1;
765
766     if (diversityConsequences.size() > 0) {
767
768       DemandFX lastConcern = diversityConsequences.get(lastIndex);
769
770       // If leader is not addressing diversity concerns
771       if (lastConcern.getState() != DemandState.COMPLETE) {
772
773         if (diversityConsequences.size() > getConsequenceThreshold()) {
774           // give the agent a chance for day off if the diversity
```

```
    consequences are getting
775       // too large
776       Random rand = new Random();
777
778       if (rand.nextBoolean()) {
779        return true;
780       }
781      }
782     }
783    }
784
785    // If leader is addressing diversity concerns
786    return false;
787   }
788
789   public boolean generateAgentDeparture() {
790    if (this.agentAbsences.get() > this.getAbsenceThreshold()) {
791     // || this.diversityConsequences.size() > this.
      getConsequenceThreshold()) {
792     return true;
793    }
794    return false;
795   }
796
797   // LEADER ACTION METHODS
798   public boolean teamNeedsSupport() {
799    agentTime.set(agentTime.get() + (180)); // 180 seconds => 5% of work
      time for checks
800
801    List<Agent> keysAsArray = new ArrayList<Agent>(this.
      agentIntDensityTable.keySet());
802    keysAsArray.remove(this);
803
804    for (Agent agent : keysAsArray) {
805
806     // With SBA, leader has insight into diversity concerns
807
808     if (this.sociometricBadgeAwareness.get()) {
809
810      if (agent.diversityConsequences.size() > 0.25 * agent.
      getConsequenceThreshold()) {
811       int last_index = agent.diversityConsequences.size() - 1;
812
813       DemandFX last_demand = agent.diversityConsequences.get(last_index);
814       if (last_demand.getState() != DemandState.COMPLETE) {
815
816        return true;
817       }
818      } else {
819
820      }
821
822      // Without SBA, assume leader only sees absences;
823      // then intervention triggers daily supports
```

```
824     } else if (!this.jointInterventionTable.isEmpty()
825       && agent.diversityConsequences.size() > 0.25 * this.
        getConsequenceThreshold()) {
826      return true;
827     } else if (agent.getAgentAbsences() > .5 * agent.getAbsenceThreshold
        ()
828        && agent.diversityConsequences.size() > 0) {
829       return true;
830     }
831    }
832    return false;
833   }
834
835   private void generateLeaderIntevention(Agent affectedAgent) {
836    /*
837     * TODO: give leader a chance to survey the environment. Are people
        taking too
838     * many days off? If so, conduct team intervention. Leader
        intervention
839     * simulates a positive action as described in the literature. Intent
        will be to
840     * assume that intervention works at first. If time is available, TODO
        : include
841     * poor leader interventions.
842     *
843     * Definitions "agent" is the agent that triggered the intervention "
        agent1" is
844     * a variable name used to iterate through the agents "this" is the
        agent with
845     * leadership skills (only .LEAD gets to this method)
846     *
847     */
848
849    // test print
850    System.out.println("---------------------INTERVENTION
        ---------------------INTERVENTION---------------------");
851    System.out.println(affectedAgent.name.get() + " Days off: " +
        affectedAgent.getAgentAbsences() + "\n");
852
853    // Assumption: 1 down day with team clears concerns of troubling agent
854    affectedAgent.diversityConsequences.stream().forEach(d -> d.setState(
        DemandState.COMPLETE));
855    this.internalDemandsMoot.addAll(affectedAgent.diversityConsequences);
856    affectedAgent.diversityConsequences.clear();
857
858    // Credit internal network density due to intervention.
859    double factor = .1;
860
861    // adjust agents' tracking tables
862    List<Agent> keysAsArray = new ArrayList<Agent>(this.
        agentIntDensityTable.keySet());
863    keysAsArray.remove(this);
864
865    for (Agent agent1 : keysAsArray) {
```

```
866    if (agent1 != this) {
867
868     for (Agent agent2 : keysAsArray) {
869      if (agent2 != this) {
870
871       double cid = agent1.agentIntDensityTable.get(agent2);
872       agent1.agentIntDensityTable.put(agent2, (Math.min((1 + factor) *
      cid, .95)));
873       agent1.jointInterventionTable.put(agent2, true);
874      }
875     }
876
877     agent1.refreshAveIntNetDensity();
878     // agent1.agentTime.set(agent1.agentTime.get() + (3600 * 8)); //
      Assumption: 1
879     // day enough to "intervene?"
880     this.jointInterventionTable.put(agent1, true);
881    }
882   }
883
884   // this.agentTime.set(agentTime.get() + (3600 * 8));
885   setNumInterventions(getNumInterventions() + 1);
886  }
887
888  private int generateMicroSupport(Agent agent) {
889   /**
890    * Leader support method. "This" agent is restricted to leaders for
      now, and
891    * "agent" is the agent the leader is supporting.
892    *
893    */
894   double cid = this.agentIntDensityTable.get(agent);
895   double factor = 1; // boost (+/-) due to interaction
896
897   // Establish a concern for leader to "support"
898   // Assumption: FIFO satisfaction of diversity consequences
899
900   DemandFX firstConcern = agent.diversityConsequences.get(0);
901
902   Random rand = new Random();
903
904   // Assumption: previous intervention ==> higher impact?
905   if (this.jointInterventionTable.containsKey(agent)) {
906    // current internal density boost
907    cid = Math.min((1 + factor) * cid, .95);
908    this.agentIntDensityTable.put(agent, cid);
909    agent.agentIntDensityTable.put(this, cid);
910    this.refreshAveIntNetDensity();
911    agent.refreshAveIntNetDensity();
912
913    // satisfy concern
914    agent.diversityConsequences.remove(firstConcern); // FIFO?
915    firstConcern.setState(DemandState.COMPLETE);
916    this.internalDemandsMoot.add(firstConcern);
```

```
917
918     // track leader progress
919     setNumMicroSupports(getNumMicroSupports() + 1);
920
921     return firstConcern.getEffort();
922
923     // No previous interaction, so micro support may/may not be helpful
924     // Assume 60% odds it's helpful
925   } else if (rand.nextInt(100) < 60) {
926
927     // current internal density boost
928     cid = Math.min((1 + factor) * cid, .95);
929     this.agentIntDensityTable.put(agent, cid);
930     agent.agentIntDensityTable.put(this, cid);
931     this.refreshAveIntNetDensity();
932     agent.refreshAveIntNetDensity();
933
934     // satisfy concern
935     agent.diversityConsequences.remove(firstConcern); // FIFO?
936     firstConcern.setState(DemandState.COMPLETE);
937     this.internalDemandsMoot.add(firstConcern);
938
939     // track leader progress
940     setNumMicroSupports(getNumMicroSupports() + 1);
941     return firstConcern.getEffort();
942
943     // 40% odds it's unhelpful TODO: add odds that it hurts?
944   } else {
945     // current internal density boost
946     this.agentIntDensityTable.put(agent, cid);
947     agent.agentIntDensityTable.put(this, cid);
948     this.refreshAveIntNetDensity();
949     agent.refreshAveIntNetDensity();
950
951     // this.jointInterventionTable.put(agent, true);
952     // agent.jointInterventionTable.put(this, true);
953
954     return 0;
955
956   }
957
958 }
959
960 // CONVENIENCE METHODS
961 class DemandPriorityComparator implements Comparator<DemandFX> {
962
963   @Override
964   public int compare(DemandFX demand1, DemandFX demand2) {
965     return Integer.compare(demand1.getPriority(), demand2.getPriority());
966   }
967
968 }
969
970 // GETTERS AND SETTERS
```

```
971 public UUID getId() {
972  return id;
973 }
974
975 public void setId(UUID id) {
976  this.id = id;
977 }
978
979 public ObjectProperty<AgentMode> getMode() {
980  return mode;
981 }
982
983 public void setMode(ObjectProperty<AgentMode> mode) {
984  this.mode = mode;
985 }
986
987 public BooleanProperty getLowRepresentation() {
988  return lowRepresentation;
989 }
990
991 public void setLowRepresentation() {
992  if (this.race.get() == Race.LOW_REP) {
993   this.lowRepresentation = new SimpleBooleanProperty(true);
994  } else {
995   this.lowRepresentation = new SimpleBooleanProperty(false);
996  }
997 }
998
999 public BooleanProperty getModerateRepresentation() {
1000  return moderateRepresentation;
1001 }
1002
1003 public void setModerateRepresentation() {
1004  if (this.lowRepresentation.get() != true && this.gender.get() ==
       Gender.LOW_REP) {
1005   this.moderateRepresentation = new SimpleBooleanProperty(true);
1006  } else {
1007   this.moderateRepresentation = new SimpleBooleanProperty(false);
1008  }
1009 }
1010
1011 public ListProperty<DemandFX> getGlobalAssignedDemands() {
1012  return internalAssignedDemands;
1013 }
1014
1015 public void setGlobalAssignedDemands(ListProperty<DemandFX>
       globalAssignedDemands) {
1016  this.internalAssignedDemands = globalAssignedDemands;
1017 }
1018
1019 public ListProperty<DemandFX> getGlobalEmergingSignals() {
1020  return internalEmergingSignals;
1021 }
1022
```

```
1023 public void setGlobalEmergingSignals(ListProperty<DemandFX>
       globalEmergingSignals) {
1024  this.internalEmergingSignals = globalEmergingSignals;
1025 }
1026
1027 public ListProperty<DemandFX> getInternalDemandsPending() {
1028  return internalDemandsPending;
1029 }
1030
1031 public void setInternalDemandsPending(ListProperty<DemandFX>
       internalDemandsPending) {
1032  this.internalDemandsPending = internalDemandsPending;
1033 }
1034
1035 public ListProperty<DemandFX> getInternalDemandsActive() {
1036  return internalDemandsActive;
1037 }
1038
1039 public void setInternalDemandsActive(ListProperty<DemandFX>
       internalDemandsActive) {
1040  this.internalDemandsActive = internalDemandsActive;
1041 }
1042
1043 public ListProperty<DemandFX> getInternalDemandsMoot() {
1044  return internalDemandsMoot;
1045 }
1046
1047 public void setInternalDemandsMoot(ListProperty<DemandFX>
       internalDemandsMoot) {
1048  this.internalDemandsMoot = internalDemandsMoot;
1049 }
1050
1051 public ListProperty<SupplyFX> getSupplies() {
1052  return supplies;
1053 }
1054
1055 public void setSupplies(ListProperty<SupplyFX> supplies) {
1056  this.supplies = supplies;
1057 }
1058
1059 private void setInitialSupplies(ObjectProperty<AgentType> _type) {
1060  if(_type.get() == AgentType.LEADER) {
1061   SupplyFX s1 = new SupplyBase(ActType.WORK, 3600 * 6, 9, 17, "WORK");
1062   SupplyFX s2 = new SupplyBase(ActType.DECIDE, 3600 * 0, 9, 17, "DECIDE
       ");
1063   SupplyFX s3 = new SupplyBase(ActType.LEAD, 3600 * 2, 9, 17, "LEAD");
1064   this.supplies.addAll(s1, s2, s3);
1065
1066  } else if (type.get() == AgentType.ENGINEER) {
1067   SupplyFX s1 = new SupplyBase(ActType.DESIGN, 3600 * 8, 9, 17, "DESIGN
       ");
1068   SupplyFX s2 = new SupplyBase(ActType.COORD, 3600 * 8, 9, 17, "COORD")
       ;
1069   supplies.addAll(s1, s2);
```

```
1070    }
1071    setSupplies(supplies);
1072  }
1073
1074  public IntegerProperty getAgentTime() {
1075    return agentTime;
1076  }
1077
1078  public void setAgentTime(IntegerProperty agentTime) {
1079    this.agentTime = agentTime;
1080  }
1081
1082  public IntegerProperty getAgentTimeActing() {
1083    return agentTimeActing;
1084  }
1085
1086  public void setAgentTimeActing(IntegerProperty agentTimeActing) {
1087    this.agentTimeActing = agentTimeActing;
1088  }
1089
1090  public IntegerProperty getTimeLimit() {
1091    return agentHorizon;
1092  }
1093
1094  public void setTimeLimit(IntegerProperty timeLimit) {
1095    this.agentHorizon = timeLimit;
1096  }
1097
1098  public IntegerProperty getTimeStep() {
1099    return timeStep;
1100  }
1101
1102  public void setTimeStep(IntegerProperty timePausePeriod) {
1103    this.timeStep = timePausePeriod;
1104  }
1105
1106  public IntegerProperty getAgentLoopDelay() {
1107    return agentLoopDelay;
1108  }
1109
1110  public void setAgentLoopDelay(IntegerProperty agentLoopDelay) {
1111    this.agentLoopDelay = agentLoopDelay;
1112  }
1113
1114  public ObjectProperty<DemandFX> getSelectedDemand() {
1115    return selectedDemand;
1116  }
1117
1118  public void setSelectedDemand(ObjectProperty<DemandFX> selectedDemand)
       {
1119    this.selectedDemand = selectedDemand;
1120  }
1121
1122  public ObjectProperty<SupplyFX> getSelectedSupply() {
```

```
1123  return selectedSupply;
1124 }
1125
1126 public void setSelectedSupply(ObjectProperty<SupplyFX> selectedSupply)
        {
1127  this.selectedSupply = selectedSupply;
1128 }
1129
1130 public ObjectProperty<ActFX> getInProcessAct() {
1131  return inProcessAct;
1132 }
1133
1134 public void setInProcessAct(ObjectProperty<ActFX> inProcessAct) {
1135  this.inProcessAct = inProcessAct;
1136 }
1137
1138 public ListProperty<ActFX> getActs() {
1139  return acts;
1140 }
1141
1142 public void setActs(ListProperty<ActFX> acts) {
1143  this.acts = acts;
1144 }
1145
1146 public final StringProperty nameProperty() {
1147  return this.name;
1148 }
1149
1150 public final String getName() {
1151  return this.nameProperty().get();
1152 }
1153
1154 public final void setName(final String name) {
1155  this.nameProperty().set(name);
1156 }
1157
1158 public final ObjectProperty<AgentType> typeProperty() {
1159  return this.type;
1160 }
1161
1162 public final AgentType getType() {
1163  return this.typeProperty().get();
1164 }
1165
1166 public final void setType(final AgentType type) {
1167  this.typeProperty().set(type);
1168 }
1169
1170 public final ObjectProperty<Race> raceProperty() {
1171  return this.race;
1172 }
1173
1174 public final Race getRace() {
1175  return this.raceProperty().get();
```

```
1176  }
1177
1178  public final void setRace(final Race race) {
1179   this.raceProperty().set(race);
1180  }
1181
1182  public final ObjectProperty<Gender> genderProperty() {
1183   return this.gender;
1184  }
1185
1186  public final Gender getGender() {
1187   return this.genderProperty().get();
1188  }
1189
1190  public final void setGender(final Gender gender) {
1191   this.genderProperty().set(gender);
1192  }
1193
1194  public final ListProperty<DemandFX> internalAssignedDemandsProperty() {
1195   return this.internalAssignedDemands;
1196  }
1197
1198  public final ObservableList<DemandFX> getInternalAssignedDemands() {
1199   return this.internalAssignedDemandsProperty().get();
1200  }
1201
1202  public final void setInternalAssignedDemands(final ObservableList<
        DemandFX> internalAssignedDemands) {
1203   this.internalAssignedDemandsProperty().set(internalAssignedDemands);
1204  }
1205
1206  public final ListProperty<DemandFX> diversityConcernsProperty() {
1207   return this.diversityConsequences;
1208  }
1209
1210  public final ObservableList<DemandFX> getDiversityConcerns() {
1211   return this.diversityConcernsProperty().get();
1212  }
1213
1214  public final void setDiversityConcerns(final ObservableList<DemandFX>
        diversityConcerns) {
1215   this.diversityConcernsProperty().set(diversityConcerns);
1216  }
1217
1218  public final ListProperty<DemandFX> internalEmergingSignalsProperty() {
1219   return this.internalEmergingSignals;
1220  }
1221
1222  public final ObservableList<DemandFX> getInternalEmergingSignals() {
1223   return this.internalEmergingSignalsProperty().get();
1224  }
1225
1226  public final void setInternalEmergingSignals(final ObservableList<
        DemandFX> internalEmergingSignals) {
```

```
1227  this.internalEmergingSignalsProperty().set(internalEmergingSignals);
1228 }
1229
1230 public final ListProperty<DemandFX> diversityConsequencesProperty() {
1231  return this.diversityConsequences;
1232 }
1233
1234 public final ObservableList<DemandFX> getDiversityConsequences() {
1235  return this.diversityConsequencesProperty().get();
1236 }
1237
1238 public final void setDiversityConsequences(final ObservableList<
       DemandFX> diversityConsequences) {
1239  this.diversityConsequencesProperty().set(diversityConsequences);
1240 }
1241
1242 public final IntegerProperty agentHorizonProperty() {
1243  return this.agentHorizon;
1244 }
1245
1246 public final int getAgentHorizon() {
1247  return this.agentHorizonProperty().get();
1248 }
1249
1250 public final void setAgentHorizon(final int agentHorizon) {
1251  this.agentHorizonProperty().set(agentHorizon);
1252 }
1253
1254 public final IntegerProperty agentAbsencesProperty() {
1255  return this.agentAbsences;
1256 }
1257
1258 public final int getAgentAbsences() {
1259  return this.agentAbsencesProperty().get();
1260 }
1261
1262 public final void setAgentAbsences(final int agentAbsences) {
1263  this.agentAbsencesProperty().set(agentAbsences);
1264 }
1265
1266 public final DoubleProperty absenceThresholdProperty() {
1267  return this.absenceThreshold;
1268 }
1269
1270 public final double getAbsenceThreshold() {
1271  return this.absenceThresholdProperty().get();
1272 }
1273
1274 public void setAbsenceThreshold(DoubleProperty absenceThreshold) {
1275  this.absenceThreshold = absenceThreshold;
1276 }
1277
1278 public BooleanProperty getTransormationalLeader() {
1279  return transformationalLeader;
```

```
1280  }
1281
1282  public void setTransormationalLeader(BooleanProperty
        transormationalLeader) {
1283   this.transformationalLeader = transormationalLeader;
1284  }
1285
1286  public final DoubleProperty consequenceThresholdProperty() {
1287   return this.consequenceThreshold;
1288  }
1289
1290  public final double getConsequenceThreshold() {
1291   return this.consequenceThresholdProperty().get();
1292  }
1293
1294  public final void setConsequenceThreshold(final double
        consequenceThreshold) {
1295   this.consequenceThresholdProperty().set(consequenceThreshold);
1296  }
1297
1298  public BooleanProperty getSociometricBadgeAwareness() {
1299   return sociometricBadgeAwareness;
1300  }
1301
1302  public void setSociometricBadgeAwareness(BooleanProperty
        sociometricBadgeAwareness) {
1303   this.sociometricBadgeAwareness = sociometricBadgeAwareness;
1304  }
1305
1306  public final ObjectProperty<DiversityApproach>
        diversityApproachProperty() {
1307   return this.diversityApproach;
1308  }
1309
1310  public final DiversityApproach getDiversityApproach() {
1311   return this.diversityApproachProperty().get();
1312  }
1313
1314  public final void setDiversityApproach(final DiversityApproach
        diversityApproach) {
1315   this.diversityApproachProperty().set(diversityApproach);
1316  }
1317
1318  public int getPostInterventionPeriod() {
1319   return postInterventionPeriod;
1320  }
1321
1322  public void setPostInterventionPeriod(int postInterventionPeriod) {
1323   this.postInterventionPeriod = postInterventionPeriod;
1324  }
1325
1326  public final BooleanProperty transformationalLeaderProperty() {
1327   return this.transformationalLeader;
1328  }
```

```
1329
1330  public final boolean isTransformationalLeader() {
1331   return this.transformationalLeaderProperty().get();
1332  }
1333
1334  public final void setTransformationalLeader(final boolean
        transformationalLeader) {
1335   this.transformationalLeaderProperty().set(transformationalLeader);
1336  }
1337
1338  public final IntegerProperty numInterventionsProperty() {
1339   return this.numInterventions;
1340  }
1341
1342  public final int getNumInterventions() {
1343   return this.numInterventionsProperty().get();
1344  }
1345
1346  public final void setNumInterventions(final int numInterventions) {
1347   this.numInterventionsProperty().set(numInterventions);
1348  }
1349
1350  public final IntegerProperty numMicroSupportsProperty() {
1351   return this.numMicroSupports;
1352  }
1353
1354  public final int getNumMicroSupports() {
1355   return this.numMicroSupportsProperty().get();
1356  }
1357
1358  public final void setNumMicroSupports(final int numMicroSupports) {
1359   this.numMicroSupportsProperty().set(numMicroSupports);
1360  }
1361
1362  public final ListProperty<Double> aveIntDensityListProperty() {
1363   return this.aveIntDensityList;
1364  }
1365
1366  public final ObservableList<Double> getAveIntDensityList() {
1367   return this.aveIntDensityListProperty().get();
1368  }
1369
1370  public final void setAveIntDensityList(final ObservableList<Double>
        aveIntDensityList) {
1371   this.aveIntDensityListProperty().set(aveIntDensityList);
1372  }
1373
1374 }
```

A.3.2 AgentMode.java

```
1
2 public enum AgentMode {
3  INACTIVE, OBSERVING, SELECTING, ACTING, FINISHING, WAITING, COMPLETING,
```

```
    PAUSED, TIMELIMITED, RECOVERING, DEPARTED,
 SUPPORTING
}
```

A.3.3 AgentType.java

```

public enum AgentType {
 LEADER, ENGINEER
}
```

A.3.4 Supply.java

```


public class Supply {

 // states
 SupplyType type;
 String name;
 int priority;
 int start;
 int stop;
 int capacity;
 boolean recur;
 int every;
 int until;

 public Supply(SupplyType _type, String _name, int _priority, int _start
    , int _stop, int _capacity, boolean _recur,
   int _every, int _until) {
  type = _type;
  name = _name;
  priority = _priority;
  start = _start;
  stop = _stop;
  capacity = _capacity;
  recur = _recur;
  every = _every;
  until = _until;
 }

 public SupplyType getType() {
  return type;
 }

 public void setType(SupplyType type) {
  this.type = type;
 }

 public String getName() {
  return name;
 }

```

```
41 public void setName(String name) {
42  this.name = name;
43 }
44
45 public int getPriority() {
46  return priority;
47 }
48
49 public void setPriority(int priority) {
50  this.priority = priority;
51 }
52
53 public int getStart() {
54  return start;
55 }
56
57 public void setStart(int start) {
58  this.start = start;
59 }
60
61 public int getStop() {
62  return stop;
63 }
64
65 public void setStop(int stop) {
66  this.stop = stop;
67 }
68
69 public int getCapacity() {
70  return capacity;
71 }
72
73 public void setCapacity(int capacity) {
74  this.capacity = capacity;
75 }
76
77 public boolean isRecur() {
78  return recur;
79 }
80
81 public void setRecur(boolean recur) {
82  this.recur = recur;
83 }
84
85 public int getEvery() {
86  return every;
87 }
88
89 public void setEvery(int every) {
90  this.every = every;
91 }
92
93 public int getUntil() {
94  return until;
```

```
 }

 public void setUntil(int until) {
  this.until = until;
 }

 @Override
 public String toString() {
  return String.format("Created Supply:  %s; Capacity/day:  %d", name,
    capacity);
 }
}
```

A.3.5 SupplyBase.java

```

import java.util.ArrayList;
import java.util.UUID;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.ObjectProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

/**
 * A Class to represent Supplies in an agent based model
 *
 * @author Bryan Moser
 *
 */
public class SupplyBase implements SupplyFX {
 private UUID id;

 private StringProperty name;
 private ObjectProperty<ActType> type;
 private ObjectProperty<SupplyState> state;
 private IntegerProperty capacity; // effort in seconds

 // Productivity: multiplier on (demand nominal effort/ supplied effort)
 // Can this supply satisfy demand with less or more effort compared to
    nominal?
 private DoubleProperty efficiency;

 // Attributes of timing could become their own class (availability or
    calendar)
 private IntegerProperty start;
 private IntegerProperty stop;
 private BooleanProperty recur;
```

```
38 private IntegerProperty every;
39 private IntegerProperty until;
40
41 // DEFAULT VALUES
42 private static ActType DEFAULT_TYPE = ActType.DESIGN;
43 private static int DEFAULT_START = 9;
44 private static int DEFAULT_STOP = 17;
45 private static int DEFAULT_CAPACITY = 3600 * 8;
46 private static double DEFAULT_EFFICIENCY = 1.0;
47 private static int DEFAULT_EVERY = 24;
48 private static int DEFAULT_UNTIL = 24 * 14;
49 private static boolean DEFAULT_RECUR = false;
50
51 public SupplyBase() {
52  this(DEFAULT_TYPE, DEFAULT_CAPACITY);
53 }
54
55 /**
56  * Constructs a supply with default state pending with _type and
      nominal
57  * capacity
58  *
59  * @param _type       the type of demand effort from set of TYPE_
      constants in
60  *                    this class
61  * @param _effortSecs a capacity for effort (for the timing)in seconds
62  */
63 public SupplyBase(ActType _type, int _capacitySecs) {
64  this(_type, _capacitySecs, DEFAULT_START, DEFAULT_STOP, _type + ":" +
      _capacitySecs / 3600);
65 }
66
67 /**
68  * Constructs a supply object with default and the following input
      values
69  *
70  * @param _type       the type of demand effort from set of TYPE_
      constants in
71  *                    this class
72  * @param _effortSecs nominal effort in seconds
73  * @param _start      the start time as int of this demand
74  * @param _stop       the stop time as int of this demand
75  * @param _name       a human readable tag to display in UX
76  */
77 public SupplyBase(ActType _type, int _capacitySecs, int _start, int
      _stop, String _name) {
78  id = UUID.randomUUID();
79  type = new SimpleObjectProperty<ActType>(_type);
80  capacity = new SimpleIntegerProperty(_capacitySecs);
81  efficiency = new SimpleDoubleProperty(DEFAULT_EFFICIENCY);
82  start = new SimpleIntegerProperty(_start);
83  stop = new SimpleIntegerProperty(_stop);
84  name = new SimpleStringProperty(_name);
85  recur = new SimpleBooleanProperty(DEFAULT_RECUR);
```

```
86   every = new SimpleIntegerProperty(DEFAULT_EVERY);
87   until = new SimpleIntegerProperty(DEFAULT_UNTIL);
88  }
89
90  /*
91   * Several convenience methods
92   */
93
94  /**
95   * Sets a recurring pattern of supply capacity (total availability).
       This
96   * implies that in each period the supply of nominal effort is
       available This
97   * does not indicate how the supply is replenished, just that it exists
98   *
99   * @param _period
100  * @param _completion
101  */
102 public void setRecurrance(int _every, int _completion) {
103  this.setRecur(true);
104  every.set(_every);
105  until.set(_completion);
106 }
107
108 @Override
109 public String toString() {
110  String retVal = "Supply- " + name.get() + " [" + start.get() + "-" +
       stop.get() + "] " + type.get() + ": "
111    + capacity.get() / 3600.0 + " hrs";
112
113  if (recur.get()) {
114   retVal += " every " + every.get() + " until " + until.get();
115  }
116  return retVal;
117 }
118
119 /**
120  * Returns true of the supply is active during the period from start to
       top
121  *
122  * @param from  an int representing the start time of the period
123  * @param until an int representing the stop time of the period
124  * @return
125  */
126 public boolean isAvailable(int _from, int _until) {
127  return false;
128 }
129
130 /**
131  * Set the active period of this demand from start to stop does now
       allow stop
132  * before start
133  *
134  * @param _start
```

```
135   * @param _stop
136   */
137  public void setPeriod(int _start, int _stop) {
138   if (_start > _stop) {
139    return;
140   }
141   start.set(_start);
142   stop.set(_stop);
143  }
144
145  public double getCapacityHrs() {
146   return capacity.get() / 3600.0;
147  }
148
149  public void setCapacityHrs(double _capacityHrs) {
150   capacity.set((int) (_capacityHrs * 3600.0));
151  }
152
153  @Override
154  public boolean isMatch(DemandFX d) {
155   // TODO just checking name matching, but could be whole type
       hierarchical match
156   return (d.getType() == type.get());
157  }
158
159  /**
160   * return the overlap of hours for this supply with Demand for whole
       expressed
161   * life of the supply and demand does not check for availability but
       assumes not
162   * allocated to other demands
163   *
164   * @param d
165   * @return the hours of overlap
166   */
167  public double getOverlapHrs(DemandFX d) {
168   double total = 0.0;
169   for (int n = start.get(); n < this.stop.get(); n++) {
170    if (n >= d.getStart() && n <= d.getStop()) {
171     total++;
172    }
173   }
174   if (this.recur.get()) {
175
176   }
177
178   return total;
179  }
180
181  /**
182   * return the overlap of hours for this supply with Demand d during the
       period
183   * from _start to _stop
184   *
```

```
185   * @param d
186   * @param _start
187   * @param _stop
188   * @return the hours of overlap
189   */
190  public double getHoursOverlapInPeriod(DemandFX d, int _start, int _stop
        ) {
191   return 0.0; // TBD!
192  }
193
194  public static ArrayList<SupplyBase> createTestSupplySet() {
195   // Set up some supplies
196
197   SupplyBase s1 = new SupplyBase(ActType.DESIGN, 3600 * 8, 9, 17, "
        worker");
198   SupplyBase s2 = new SupplyBase(ActType.COORD, 3600 * 2, 9, 13, "talker
        ");
199
200   // Set up a demand table
201   ArrayList<SupplyBase> oList = new ArrayList<SupplyBase>();
202
203   oList.add(s1);
204   oList.add(s2);
205
206   // do various things to show and test the list
207
208   // print the Supplies in the list
209   oList.forEach(System.out::println);
210
211   return oList;
212  }
213
214  /*
215   * Generated getters and setters, including properties
216   */
217  @Override
218  public UUID getId() {
219   return id;
220  }
221
222  @Override
223  public void setId(UUID id) {
224   this.id = id;
225  }
226
227  @Override
228  public StringProperty nameProperty() {
229   return this.name;
230  }
231
232  @Override
233  public String getName() {
234   return this.nameProperty().get();
235  }
```

```
236
237 @Override
238 public void setName(String name) {
239  this.nameProperty().set(name);
240 }
241
242 @Override
243 public IntegerProperty capacityProperty() {
244  return this.capacity;
245 }
246
247 @Override
248 public int getCapacity() {
249  return this.capacityProperty().get();
250 }
251
252 @Override
253 public void setCapacity(int capacity) {
254  this.capacityProperty().set(capacity);
255 }
256
257 public DoubleProperty productivityProperty() {
258  return this.efficiency;
259 }
260
261 public double getProductivity() {
262  return this.productivityProperty().get();
263 }
264
265 public void setProductivity(double productivity) {
266  this.productivityProperty().set(productivity);
267 }
268
269 @Override
270 public IntegerProperty startProperty() {
271  return this.start;
272 }
273
274 @Override
275 public int getStart() {
276  return this.startProperty().get();
277 }
278
279 @Override
280 public void setStart(int start) {
281  this.startProperty().set(start);
282 }
283
284 @Override
285 public IntegerProperty stopProperty() {
286  return this.stop;
287 }
288
289 @Override
```

```
290 public int getStop() {
291  return this.stopProperty().get();
292 }
293
294 @Override
295 public void setStop(int stop) {
296  this.stopProperty().set(stop);
297 }
298
299 @Override
300 public BooleanProperty recurProperty() {
301  return this.recur;
302 }
303
304 @Override
305 public boolean isRecur() {
306  return this.recurProperty().get();
307 }
308
309 @Override
310 public void setRecur(boolean recur) {
311  this.recurProperty().set(recur);
312 }
313
314 @Override
315 public IntegerProperty everyProperty() {
316  return this.every;
317 }
318
319 @Override
320 public int getEvery() {
321  return this.everyProperty().get();
322 }
323
324 @Override
325 public void setEvery(int every) {
326  this.everyProperty().set(every);
327 }
328
329 @Override
330 public IntegerProperty untilProperty() {
331  return this.until;
332 }
333
334 @Override
335 public int getUntil() {
336  return this.untilProperty().get();
337 }
338
339 @Override
340 public void setUntil(int until) {
341  this.untilProperty().set(until);
342 }
343
```

```
344 @Override
345 public ObjectProperty<ActType> typeProperty() {
346  return this.type;
347 }
348
349 @Override
350 public ActType getType() {
351  return this.typeProperty().get();
352 }
353
354 @Override
355 public void setType(final ActType type) {
356  this.typeProperty().set(type);
357 }
358
359 public ObjectProperty<SupplyState> stateProperty() {
360  return this.state;
361 }
362
363 public SupplyState getState() {
364  return this.stateProperty().get();
365 }
366
367 public void setState(final SupplyState state) {
368  this.stateProperty().set(state);
369 }
370
371 @Override
372 public DoubleProperty efficiencyProperty() {
373  return efficiency;
374 }
375
376 @Override
377 public double getEfficiency() {
378  return efficiency.get();
379 }
380
381 @Override
382 public void setEfficiency(double eff) {
383  efficiency.set(eff);
384
385 }
386
387 /**
388  * In this simulator, this is the lowest level where supply meets
      demand Mutates
389  * the act with effort, duration, and success
390  */
391 @Override
392 public void attempt(ActFX a) {
393
394  DemandFX d = a._demand;
395  // assumed this fit was checked during selection, but...
396
```

```
397   if (!isMatch(d)) {
398    a._success = false;
399    return;
400   }
401
402   a._effort = (int) (d.getEffort() / this.getEfficiency());
403   a._timeStop = a._timeStart + a._effort; // minus the breaks
404   a._success = true;
405
406  }
407
408 }
```

A.3.6 SupplyFX.java

```
1
2  import java.util.UUID;
3
4  import javafx.beans.property.BooleanProperty;
5  import javafx.beans.property.DoubleProperty;
6  import javafx.beans.property.IntegerProperty;
7  import javafx.beans.property.ObjectProperty;
8  import javafx.beans.property.StringProperty;
9
10 public interface SupplyFX {
11
12  /*
13   * Generated getters and setters, including properties
14   */
15  public UUID getId();
16
17  public void setId(UUID id);
18
19  public StringProperty nameProperty();
20
21  public String getName();
22
23  public void setName(String name);
24
25  public IntegerProperty capacityProperty();
26
27  public int getCapacity();
28
29  public void setCapacity(int capacity);
30
31  // TYPE ------------------------------------
32  public ObjectProperty<ActType> typeProperty();
33
34  public ActType getType();
35
36  public void setType(ActType type);
37
38  public DoubleProperty efficiencyProperty();
39
```

```
40  public double getEfficiency();
41
42  public void setEfficiency(double eff);
43
44  public IntegerProperty startProperty();
45
46  public int getStart();
47
48  public void setStart(int start);
49
50  public IntegerProperty stopProperty();
51
52  public int getStop();
53
54  public void setStop(int stop);
55
56  public BooleanProperty recurProperty();
57
58  public boolean isRecur();
59
60  public void setRecur(boolean recur);
61
62  public IntegerProperty everyProperty();
63
64  public int getEvery();
65
66  public void setEvery(int every);
67
68  public IntegerProperty untilProperty();
69
70  public int getUntil();
71
72  public void setUntil(int until);
73
74  public boolean isMatch(DemandFX d);
75
76  public void attempt(ActFX a);
77
78  @Override
79  public String toString();
80 }
```

A.3.7 ActFX.java

```
1
2
3 public class ActFX {
4  public Agent _agent;
5  public SupplyFX _supply;
6  public DemandFX _demand;
7  public int _timeStart;
8  public int _timeStop;
9  public int _effort;
10 public boolean _success;
```

```
11
12  /**
13   * Convenience method to record the acting start attributes of an act
14   *
15   * @param timeStart
16   * @param agentFX
17   * @param s
18   * @param d
19   */
20  public void GetSet(int timeStart, Agent agentFX, SupplyFX s, DemandFX d
       ) {
21   _agent = agentFX;
22   _supply = s;
23   _demand = d;
24   _timeStart = timeStart;
25  }
26
27  /**
28   * Convenience method to record the finishing step attributes of an act
29   *
30   * @param timeStop
31   * @param effortHrs
32   * @param success
33   */
34  public void WrapUp(int timeStop, int effort, boolean success) {
35   _timeStop = timeStop;
36   _effort = effort;
37   _success = success;
38  }
39
40  @Override
41  public String toString() {
42   String retVal = "Agent - " + _agent.name.get() + " [" + _supply.
       toString() + "-" + _demand.toString() + "] ";
43
44   return retVal;
45  }
46 }
```

A.3.8 ActType.java

```
1
2 public enum ActType {
3  COORD, DECIDE, DESIGN, TRAVEL, ADMIN, WORK, LEAD
4 }
```

A.3.9 Race.java

```
1 import java.util.Random;
2
3 public enum Race {
4  LOW_REP, HIGH_REP; // NOTE - THIS WILL NEED TO BE EXPANDED WITH FURTHER
       RESEARCH
5
```

```
 public static Race getRandomRace() {
  Random random = new Random();
  if (random.nextInt(1000) <= 100) {
   return values()[0];
  } else {
   return values()[1];
  }

 }
}
```

A.3.10 Gender.java

```
import java.util.Random;

public enum Gender {
 LOW_REP, HIGH_REP; // NOTE - THIS WILL NEED TO BE EXPANDED WITH FURTHER
    RESEARCH

 public static Gender getRandomGender() {
  Random random = new Random();
  if (random.nextInt(1000) <= 400) {
   return values()[0];
  } else {
   return values()[1];
  }
 }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

Teresa M. Amabile. *The Social Psychology of Creativity*. Springer-Verlag, New York, 1983.

Teresa M. Amabile. A model of creativity and innovation in organizations. *Research in Organizational Behavior*, 10:123–167, 1988.

Teresa M. Amabile, Elizabeth A. Schatzel, Giovanni B. Moneta, and Steven J. Kramer. Leader behaviors and the work environment for creativity: Perceived leader support. *The Leadership Quarterly*, 15(1):5–32, 2004.

Deborah Gladstein Ancona and David F. Caldwell. Demography and Design : Predictors of New Product Team Performance Author (s): Deborah Gladstein Ancona and David F . Caldwell Source : Organization Science , Vol . 3 , No . 3 , Focused Issue : Management of Technology Published by : INFORMS Stable URL. *Organization Science*, 3(3):321–341, 1992. ISSN 0003-9993.

Evan P. Apfelbaum, Nicole M. Stephens, and Ray E. Reagans. Beyond one-size-fits-all: Tailoring diversity approaches to the representation of social groups. *Journal of Personality and Social Psychology*, 111(4):547–566, 2016. ISSN 00223514. doi: 10.1037/pspi0000071.

Bernard M Bass. *Leadership and Performance Beyond Expectations*. Free Press, New York, 1985.

Bernard M Bass. Two Decades of Research and Development in Transformational Leadership. *European Journal of Work and Organizational Psychology*, 8(1):9–32, 1999. ISSN 1359432X.

Bernard M Bass and Bruce J. Avolio. *Transformational Leadership Development: Manual for the Multifactor Leadership Questionnaire*. Consulting Psychologists Press, Palot Alto, CA, 1990.

Suzanne T. Bell, Anton J. Villado, Marc A. Lukasik, Larisa Belau, and Andrea L. Briggs. Getting specific about demographic diversity variable and team performance relationships: A meta-analysis. *Journal of Management*, 37(3):709–743, 2011. ISSN 01492063. doi: 10.1177/0149206310365001.

Marilynn B. Brewer. The psychology of prejudice: Ingroup love or outgroup hate? *Journal of Social Issues*, 55(3):429–444, 1999. ISSN 00224537. doi: 10.1111/0022-4537.00126.

James M. Burns. *Leadership*. Harper & Row, New York, 1978.

Ashli B. Carter and Katherine W. Phillips. The double-edged sword of diversity: Toward a dual pathway model. *Social and Personality Psychology Compass*, 11(5): 1–14, 2017. ISSN 17519004. doi: 10.1111/spc3.12313.

Jennifer A. Chatman, Jeffrey T. Polzer, Sigal G. Barsade, and Margaret A. Neale. Being different yet feeling similar: The influence of demographic composition and organizational culture on work processes and outcomes. *Administrative Science Quarterly*, 43(4):749–780, 1998. ISSN 00018392. doi: 10.2307/2393615.

Susan Cohen and Diane Bailey. What makes teams work: Group effectiveness research from the shop floor to the executive suite. *Journal of Management*, 23(3):239–290, 1997. URL https://doi.org/10.1016/S0149-2063(97)90034-9.

Taylor H. Cox, Sharon A. Lobel, and Poppy Lauretta McLeod. Effects of Ethnic Group Cultural Differences on Cooperative and Competitive Behavior On a Group Task. *Academy of Management Journal*, 34(4):827–847, 1991. ISSN 0001-4273. doi: 10.5465/256391.

Carlos Ricardo De Abreu Dos Reis, Miguel Ángel Sastre Castillo, and Salvador Roig Dobón. Diversity and business performance: 50 years of research. *Service Business*, 1(4):257–274, 2007. ISSN 18628516. doi: 10.1007/s11628-007-0022-2.

Bart De Jong, Kurt Dirks, and Nicole Gillespie. Trust and team performance: A meta-Analysis of main effects, contingencies, and qualifiers. *75th Annual Meeting of the Academy of Management, AOM 2015*, 101(8):744–749, 2015. ISSN 0065-0668. doi: 10.5465/AMBPP.2015.234.

P. C. Earley. East Meets West Meets Mideast: Further Explorations of Collectivistic and Individualistic Work Groups. *Academy of Management Proceedings*, 1991(1): 205–209, 1991. ISSN 0065-0668. doi: 10.5465/ambpp.1991.4976900.

Julio A Espinoza and Raymond T. Garza. Social group salience and inter-ethnic cooperation. *Journal of Experimental Social Psychology*, 21:380–392, 1985. doi: 10.1016/0022-1031(85)90037-X.

Samuel L. Gaertner, Jeffrey A. Mann, John F. Dovidio, Audrey J. Murrell, and Marina Pomare. How Does Cooperation Reduce Intergroup Bias? *Journal of Personality and Social Psychology*, 59(4):692–704, 1990. ISSN 00223514. doi: 10.1037/0022-3514.59.4.692.

Lale Gumusluoglu and Arzu Ilsev. Transformational leadership, creativity, and organizational innovation. *Journal of Business Research*, 62(4):461–473, 2009. ISSN

01482963. doi: 10.1016/j.jbusres.2007.07.032. URL http://dx.doi.org/10.1016/j.jbusres.2007.07.032.

J. Richard Hackman. The Design of Work Teams. In JW Lorsch, editor, *Handbook of Organizational Behavior*, pages 315–342. Prentice-Hall Harrison, Englewood Cliffs, NJ, 1987.

J. Richard Hackman and Charles G. Morris. Group tasks, group interaction process, and group performance effectiveness: A review and proposed integration. *Advances in Experimental Social Psychology*, 8:45–99, 1975. URL https://doi.org/10.1016/S0065-2601(08)60248-8.

David A. Harrison, Kenneth H. Price, and Myrtle P. Bell. Beyond relational demography: Time and the effects of surface- and deep-level diversity on work group cohesion. *Academy of Management Journal*, 41(1):96–107, 1998. ISSN 00014273. doi: 10.2307/256901.

L. R. Hoffman, E. Harburg, and N. R.F. Maier. Differences and disagreement as factors in creative group problem solving. *Journal of Abnormal and Social Psychology*, 64(3):206–214, 1962. ISSN 0096851X. doi: 10.1037/h0045952.

Sujin K. Horwitz and Irwin B. Horwitz. The effects of team diversity on team outcomes: A meta-analytic review of team demography. *Journal of Management*, 33 (6):987–1015, 2007. ISSN 01492063. doi: 10.1177/0149206307308587.

Daniel R. Ilgen, John R. Hollenbeck, Michael Johnson, and Dustin Jundt. Teams in organizations: From input-process-output models to IMOI models. *Annual Review of Psychology*, 56:517–543, 2005. ISSN 00664308. doi: 10.1146/annurev.psych.56.091103.070250.

Dong I. Jung. Transformational and transactional leadership and their effects on creativity in groups. *Creativity Research Journal*, 13(2):185–195, 2001. ISSN 10400419. doi: 10.1207/s15326934crj1302_6.

R.M. Kanter. *Men and Women of the Corporation*. Basic Books, New York, NY, 1977.

Eric Kearney and Diether Gebert. Managing Diversity and Enhancing Team Outcomes: The Promise of Transformational Leadership. *Journal of Applied Psychology*, 94(1):77–89, 2009. ISSN 00219010. doi: 10.1037/a0013077.

Moonjoo Kim. Effects of team diversity, transformational leadership, and perceived organizational support on team-learning behavior. *Social Behavior and Personality*, 45(8):1255–1270, 2017. ISSN 03012212. doi: 10.2224/sbp.6325.

Steve W.J. Kozlowski and Georgia T. Chao. Unpacking team process dynamics and emergent phenomena: Challenges, conceptual advances, and innovative methods. *American Psychologist*, 73(4):576–592, 2018. ISSN 0003066X. doi: 10.1037/amp0000245.

Steve W.J. Kozlowski and Daniel R. Ilgen. Enhancing the Effectiveness of Work Groups and Teams: A Reflection. *Psychological Science in the Public Interest*, 7 (3):77–124, 2006. doi: 10.1111/j.1529-1006.2006.00030.x.

R. Kramer. Intergroup relations and organizational dilemmas: The role of categorization processes. *Research in organizational behavior*, 13:191–228, 1991.

John M. Levine and Richard L. Moreland. Progress in small group research. *Annual Review of Psychology*, 41(1):585–634, 1990. ISSN 00664308. doi: 10.1146/annurev.ps.41.020190.003101.

Michelle A. Marks, John E. Mathieu, and Stephen J. Zaccaro. A temporally based framework and taxonomy of team processes. *Academy of Management Review*, 26 (3):356–376, 2001. ISSN 03637425. doi: 10.5465/AMR.2001.4845785.

John Mathieu, Travis M. Maynard, Tammy Rapp, and Lucy Gilson. Team effectiveness 1997-2007: A review of recent advancements and a glimpse into the future. *Journal of Management*, 34(3):410–476, 2008. ISSN 01492063. doi: 10.1177/0149206308316061.

John E Mathieu, John R Hollenbeck, and Daniel R Ilgen. Supplemental Material for A Century of Work Teams in the Journal of Applied Psychology. *Journal of Applied Psychology*, 102(3):452–467, 2017. ISSN 0021-9010. doi: 10.1037/apl0000128.supp.

John E. Mathieu, Mikhail A. Wolfson, and Semin Park. The evolution of work team research since Hawthorne. *American Psychologist*, 73(4):308–321, 2018. ISSN 0003066X. doi: 10.1037/amp0000255.

John E. Mathieu, Peter T. Gallagher, Monique A. Domingo, and Elizabeth A. Klock. Embracing Complexity: Reviewing the Past Decade of Team Effectiveness Research. *Annual Review of Organizational Psychology and Organizational Behavior*, 6:17–46, 2019. ISSN 23270608. doi: 10.1146/annurev-orgpsych-012218-015106.

McGrath. Leadership behavior: Some requirements for leadership training. Technical report, US Civil Service Commission, 1962.

Joseph E. McGrath. *Social psychology: A brief introduction.* Holt, Rinehart and Winston, New York, NY, 1964.

Joseph E. McGrath. *Groups: Interaction and Performance.* Prentice-Hall, Inc, Englewood Cliffs, NJ, 1984. ISBN 0-13-365700-0.

Joseph E. McGrath, Holly Arrow, and Jennifer L. Berdahl. The study of groups: Past, present, and future. *Personality and Social Psychology Review*, 4(1):95–105, 2000. ISSN 10888683. doi: 10.1207/S15327957PSPR0401_8.

Frances J. Milliken and Luis L. Martins. Searching for common threads: Understanding the multiple effects of diversity in organizational groups. *Academy of Management Review*, 21(2):402–433, 1996. ISSN 03637425. doi: 10.5465/AMR.1996.9605060217.

Frederick P. Morgeson, D. Scott DeRue, and Elizabeth P. Karam. *Leadership in teams: A functional approach to understanding leadership structures and processes*, volume 36. 2010. ISBN 0149206309347. doi: 10.1177/0149206309347376.

Brian Mullen and Carolyn Copper. The Relation Between Group Cohesiveness and Performance: An Integration. *Psychological Bulletin*, 115(2):210–227, 1994. ISSN 00332909. doi: 10.1037//0033-2909.115.2.210.

Lisa H. Nishii. The benefits of climate for inclusion for gender-diverse groups. *Academy of Management Journal*, 56(6):1754–1774, 2013. ISSN 00014273. doi: 10.5465/amj.2009.0823.

NRC. *Behavioral Modeling and Simulation: From Individuals to Societies*. The National Academies Press, Washington, DC, 2008. URL www.nap.edu.

Greg Oldham and Anne Cummings. Employee Creativity : Personal and Contextual Factors at Work. 39(3):607–634, 1996.

Charles A O'Reilly and Jennifer A Chatman. *Culture as social control: Corporations, cults, and commitment*, volume 18. 1996. ISBN 1-55938-938-9.

Charles A O'Reilly III, Richard C. Snyder, and Joan N. Boothe. Effects of executive team demography on organizational change. In G Huber and W Glick, editors, *Organizational Change and redesign*, chapter 5, pages 147–175. Oxford University Press, New York, NY, 1993.

Charles A O'Reilly III, Katherine Y Williams, and Sigal G. Barsade. Group demography and innovation: Does diversity help? In D. H. Gruenfeld, editor, *Research on managing groups and teams*, pages 183–207. Elsevier Science/JAI Press, 1998.

Scott E. Page. *The Diversity Bonus: How Great Teams Pay Off in the Knowledge Economy*. Princeton University Press, Princeton and Oxford, 2017. ISBN 978-0-691-19153-9.

Lisa Hope Pelled, Kathleen M Eisenhardt, and Katherine R Xin. Pelled, L.H.; Eisenhardt, K.M.; Xin, K.R. [1999] Exploring the black box_An analysis of work group diversity, conflict, and performance..pdf. *Administrative Science Quarterly*, 44(1): 1–28, 1999. ISSN 00018392.

Jeffrey Pfeffer. Organizational Demography: Implications for Management. *California Management Review*, 28(1):67–81, 1985. ISSN 21628564. doi: 10.2307/41165170.

Katherine W. Phillips, Katie A. Liljenquist, and Margaret A. Neale. Is the pain worth the gain? the advantages and liabilities of agreeing with socially distinct newcomers. *Personality and Social Psychology Bulletin*, 35(3):336–350, 2009. ISSN 01461672. doi: 10.1177/0146167208328062.

Ray Reagans, Ezra Zuckerman, and Bill McEvily. How to make the team: Social networks vs. demography as criteria for designing effective teams. *Administrative Science Quarterly*, 49(1):101–133, 2004. ISSN 00018392. doi: 10.2307/4131457.

C.M. Riordan and L. M. Shore. Demographic Diversity and Employee Attitudes. *Journal of Applied Psychology*, 82(3):342–358, 1997.

Boas Shamir. Human Relations. *Human Relations1*, 43:351–361, 1990.

M. Sherif. *The psychology of norms*. Harper & Row, New York, NY, 1936.

Shung J. Shin and Jing Zhou. When Is Educational Specialization Heterogeneity Related to Creativity in Research and Development Teams? Transformational Leadership as a Moderator. *Journal of Applied Psychology*, 92(6):1709–1721, 2007. ISSN 00219010. doi: 10.1037/0021-9010.92.6.1709.

SHRM. The Journey to Equity and Inclusion. Technical report, Society of Human Resource Management, 2020. URL https://shrmtogether.wpengine.com/wp-content/uploads/2020/08/20-1412{_}TFAW{_}Report{_}RND7{_}Pages.pdf.

Samuel R. Sommers. On racial diversity and group decision making: Identifying multiple effects of racial composition on jury deliberations. *Journal of Personality and Social Psychology*, 90(4):597–612, 2006. ISSN 00223514. doi: 10.1037/0022-3514.90.4.597.

ID Steiner. Group Process and Productivity, 1972.

Walter G. Stephan and Cookie White Stephan. Intergroup Anxiety. *Journal of Social Issues*, 41(3):157–175, 1985. ISSN 15404560. doi: 10.1111/j.1540-4560.1985.tb01134.x.

R.M. Stogdill. *Handbook of leadership: A survey of the literature*. Free Press, New York, 1974.

Amanda L. Thayer, Alexandra Petruzzelli, and Caitlin E. McClurg. Addressing the paradox of the team innovation process: A review and practical considerations. *American Psychologist*, 73(4):363–375, 2018. ISSN 0003066X. doi: 10.1037/amp0000310.

D Thomas and R Ely. Making differences matter: A new paradigm for managing diversity. *Harvard Business Review*, 74(5):79–90, 1996. ISSN 00178012. doi: 10.1225/96510.

Pamela Tierney, Steven M. Farmer, and George B. Graen. An examination of leadership and employee creativity: The relevance of traits and relationships. *Personnel Psychology*, 52(3):591–620, 1999. ISSN 00315826. doi: 10.1111/j.1744-6570.1999.tb00173.x.

J. B. Tracey and T. R. Hinkin. Leadership or managerial practices ? *Group & Organization Studies*, 23(3):220–236, 1998.

A. S. Tsui, T. Egan, and C. O'Reilly. Being Different: Relational Demography and Organizational Attachment. *Administrative Science Quarterly*, 37(4):549–579, 1992. ISSN 0065-0668. doi: 10.5465/ambpp.1991.4976867.

Anne S. Tsui and Charles A. O'reilly. Beyond Simple Demographic Effects: The Importance of Relational Demography in Superior-Subordinate Dyads. *Academy of Management Journal*, 32(2):402–423, 1989. ISSN 0001-4273. doi: 10.5465/256368.

Daan van Knippenberg. Social identity and persuasion: Reconsidering the role of group membership. In D. Abrams and M.A. Hogg, editors, *Social Identity and Social Cognition*, pages 315–331. Blackwell, Oxford, England, 1999.

Daan Van Knippenberg, Carsten K.W. De Dreu, and Astrid C. Homan. Work group diversity and group performance: An integrative model and research agenda. *Journal of Applied Psychology*, 89(6):1008–1022, 2004. ISSN 00219010. doi: 10.1037/0021-9010.89.6.1008.

Benjamin Nathan Waber. Understanding the link between changes in social support and changes in outcomes with the sociometric badge. *Dissertation Abstracts International Section A: Humanities and Social Sciences*, 73(3-A): 1100, 2012. URL http://search.ebscohost.com/login.aspx?direct=true{&}db=psyh{&}AN=2012-99171-101{&}site=ehost-live.

Warren E Watson, Kamalesh Kumar, and Larry K Michaelson. Cultural Diversity's Impact on Interaction Process and Performance : Comparing Homogeneous and Diverse Task Groups Author (s): Warren E . Watson , Kamalesh Kumar and Larry K . Michaelsen Published by : Academy of Management Stable URL : http://www.jst. *Academy of Management Journal*, 3(3):590–602, 1993.

Amy S. Wharton and James N. Baron. So Happy Together ? The Impact of Gender Segregation on Men at Work. *American Sociological Review*, 52(5):574–587, 1987.

Katherine Y Williams and Charles A O'Reilly III. Demography and Diversity in Organizations: A Review of 40 Years of Research. *Research in organizational behavior*, 20(January 1998):77, 1998. ISSN 01913085. doi: 10.1177/104649640003100505.

Gary A. Yukl and R. Lepsinger. An integrating taxonomy of managerial behavior: Implications for improving managerial effectiveness. In J.W. Jones, B.D. Steffy, and D.W. Bray, editors, *Applying Psychology in Business*. Lexington Press, Lexington, MA, 1989.

Gary A. Yukl and David D. Van Fleet. Theory and research on leadership in organizations. In M.D. Dunette and L.M. Hough, editors, *Handbook of Industrial and Organizational Psychology*, pages 147–197. Consulting Psychologists Press, Palo Alto, CA, 2, vol 3 edition, 1992.