Agent Monitoring and Mouse Interactivity in StarLogo Nova

by

Abigail Choe
S.B., C.S. M.I.T., 2017

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology

February 2019

© 2018 Massachusetts Institute of Technology. All rights reserved

Author: _________________________________________________________________

Department of Electrical Engineering and Computer Science
December 13, 2018

Certified by: _____________________________________________________________

Eric Klopfer
Professor and Director of the MIT Scheller Teacher Education Program
Thesis Supervisor

Accepted by: _____________________________________________________________

Katrina LaCurts
Chairman, Masters of Engineering Thesis Committee
Agent Monitoring and Mouse Interactivity in StarLogo Nova

by

Abigail Choe
S.B., C.S. M.I.T., 2017

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

StarLogo Nova is a powerful programming environment that allows anyone to easily create a 3D simulation and 3D game using block-based programming language. Due to its intuitive interface and ability to model complex systems, StarLogo is mainly used in primary and secondary school class settings. While StarLogo Nova effectively visualizes a wide range of simulations from biology to economy, the current implementation has two areas for improvement that will help cover more use cases. First, StarLogo Nova currently offers statistics only at the level of agent types. Users can track, for example, changes in population of a given agent type, but cannot track property changes of an individual agent. Second, users cannot interact with individual agents using the mouse. Mouse click enables user to perform targeted interaction with the simulation by triggering an event at the time and location of his or her choice. This thesis outlines agent monitoring feature and design of mouse interactivity that enable users to inspect fine details of a simulation and provide students with higher educational values.

Thesis Supervisor: Eric Klopfer
Title: Professor and Director of the MIT Scheller Teacher Education Program
Acknowledgements

First of all, I would like to thank my thesis faculty supervisor, Eric Klopfer, for providing the necessary support during the M.Eng. program and completion of my thesis. His passion for StarLogo and education made me more motivated for my work, and his leadership and humor made the lab a great place to work at.

I would also like to thank my direct supervisor, Daniel Wendel. He is a great mentor who not only provides feedback and advice on the project, but also cares for all M.Eng. students working with him. His support and encouragement provided great help through difficult times and enabled me to complete the M.Eng program.

Lastly, I am grateful to my friends and family who were always by my side and showed constant support and faith in me. This would not have been possible without them. I send my biggest love and gratitude to them.
# Table of Contents

1 Introduction 7

1.1 What is StarLogo Nova 9

1.2 Layout of StarLogo Nova 10

1.3 Educational Value of StarLogo 12

1.4 Problem Definition 12

2 Background 15

2.1 Common Use Cases 15

2.1.1 Geometry Simulation 15

2.1.2 Diffusion Simulation 16

2.1.3 Epidemiology Simulation 17

2.1.4 Ecology Simulation 19

2.2 Motivation 21

3 Agent Monitoring 22

3.1 Agent Monitoring in StarLogo TNG 22

3.2 Current Workarounds for Agent Monitoring in StarLogo Nova 23

3.2.1 Method 1: Create a separate agent type 23

3.2.2 Method 2: Set a special trait 24

3.2.3 Limitations 24

4 User Interface 26

4.1 Heuristics for User Interface 26

4.1.1 Learnability 26

4.1.2 Efficiency 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.3 Safety</td>
<td>27</td>
</tr>
<tr>
<td>4.2 UI Design for Agent Monitoring Pop-up</td>
<td>27</td>
</tr>
<tr>
<td>5 Technical Approach</td>
<td>30</td>
</tr>
<tr>
<td>5.1 Ray Casting</td>
<td>30</td>
</tr>
<tr>
<td>5.2 Color Picking</td>
<td>32</td>
</tr>
<tr>
<td>5.3 Implementing Color Picking</td>
<td>33</td>
</tr>
<tr>
<td>5.3.1 Intermediate Step for Debugging</td>
<td>33</td>
</tr>
<tr>
<td>5.3.2 Agent Click Detection</td>
<td>34</td>
</tr>
<tr>
<td>5.4 Implementing Agent Monitoring Pop-up</td>
<td>36</td>
</tr>
<tr>
<td>5.5 Supported Features</td>
<td>40</td>
</tr>
<tr>
<td>6 Design for Extending Mouse Interactivity</td>
<td>41</td>
</tr>
<tr>
<td>6.1 Significant Use Cases</td>
<td>41</td>
</tr>
<tr>
<td>6.1.1 Custom Click, Double Click, and Drag Events</td>
<td>41</td>
</tr>
<tr>
<td>6.1.2 Game Control</td>
<td>42</td>
</tr>
<tr>
<td>6.2 Design of Mouse Blocks</td>
<td>44</td>
</tr>
<tr>
<td>6.3 UI Design Principles for Mouse Blocks</td>
<td>47</td>
</tr>
<tr>
<td>7 Future Work</td>
<td>49</td>
</tr>
<tr>
<td>8 Conclusion</td>
<td>50</td>
</tr>
</tbody>
</table>
List of Figures

1-1  Simulation graphics portion of StarLogo Nova programming interface  
1-2  Blocks programming portion of StarLogo Nova interface  
1-3  An example of StarLogo Nova programming interface  
1-4  Dropdown menu for different block types  
2-1  A line graph that shows change in population in healthy agents and sick agents over time  
2-2  A line graph that shows change in population in prey and predator agents over time  
3-1  Agent monitor pop-up box in StarLogo TNG  
4-1  Proposed new design for agent monitoring pop-up  
5-1  Diagram for ray casting  
5-2  Diagram for color picking  
6-1  Different effects of left and right keyboard navigation on bird’s-eye view and first-person point of view  
6-2  New blocks in “Detection” drawer  
6-3  A new block in “Traits” drawer  
6-4  New blocks in “Mouse” drawer with dropdown menu  
6-5  New numeric blocks in “Mouse” drawer with dropdown menu  
6-6  Keyboard blocks in StarLogo Nova
1 Introduction

StarLogo Nova is an online programming environment with block-based programming language. Through StarLogo Nova, even users without previous programming experience can create a 3D game or complex simulation with over 10,000 agents simply by defining behaviours of each agent type. StarLogo Nova can teach computational thinking through block programming, as well as demonstrate topics outside of computer science through simulations of complex systems ranging from diffusion to ecology.

In addition to displaying simulation graphics, StarLogo Nova provides users with tools to look inside the simulation, namely a line graph. Visual graphics interface combined with concrete statistics allow users to make both qualitative and quantitative observations on the simulation and gain deeper understanding on the topic.

While StarLogo Nova is great for creating a big simulation with over thousands of agents and observing overall trends and changes in the simulation, it is currently not suitable for tracking the behaviour of a single agent. Ability to monitor individual agents by clicking on them, however, is crucial for understanding the relationship between the behaviour of an individual and the overall phenomenon. Moreover, ability to monitor and manipulate individual agents enable user to perform dynamic, targeted interactions during the simulation by providing direct inputs.

Agent monitoring through mouse click is the first step towards mouse interactivity. StarLogo Nova can be expanded to include mouse blocks that allow users to define custom actions at the
event of a mouse click or drag, instead of automatically displaying an agent monitoring pop-up.

This thesis project outlines implementation of agent monitoring through mouse click in StarLogo Nova and describes design for extending mouse interactivity to support custom mouse event handler.

Figure 1-1 Simulation graphics portion of StarLogo Nova programming interface. Users can use “Edit Widgets” to create buttons, toggles, etc. such as “setup,” “play,” and “score” widgets shown in the figure. Created widgets can be used in programming to control the simulation. “Edit Breeds” allows users to add agent types, and remaining four buttons control camera view of the simulation.
1.1 What is StarLogo Nova

StarLogo Nova is an online programming environment developed by the MIT STEP (Scheller Teacher Education Program) that allows anyone to create a 3D game or simulation using easy-to-understand block-based, agent-oriented programming interface [9, 10]. It is a new online iteration of StarLogo TNG (The Next Generation), a downloadable programming environment. Contrary to StarLogo TNG, StarLogo Nova does not require any installation and allows users to share projects more easily.

Figure 1-2 Blocks programming portion of StarLogo Nova interface. On the left is a navigation dropdown menu that displays blocks under selected drawer. Users can drag and drop blocks from the drawer onto the page.
1.2 Layout of StarLogo Nova

StarLogo Nova project page consists of two main portions. On top, users see 3D simulation graphics with control widgets (Figure 1-1). Users can click “Edit Widgets” button on top-left corner to add more widgets including buttons, toggles, and labels, such as “setup,” “play,” and “score” widgets shown in Figure 1-1. These widgets are reflected in programming portion and can be used to control the simulation, trigger events, and display system properties.

Scrolling down from simulation graphics leads to blocks programming portion of StarLogo Nova (Figure 1-2). On the left, users can navigate through different drawers that contain relevant blocks. Once users find an appropriate block, the block can simply be dragged and dropped onto the page. When users press the “Run Code” button, the block code is compiled and resulting simulation is rendered.

![Figure 1-3 An example of StarLogo Nova programming interface.](image)

On top, there are three tabs: “Turtle,” “Everyone,” and “The World.” “The World” refers to the entire simulation world. “Turtle” is a type of agent, and “Everyone” encompasses all agent types. In this example, since there is only one agent type, “Everyone” includes only “Turtle.” The block of code defined in “Turtle” tab is applied to all “Turtle” agents created.
The programming interface of StarLogo Nova uses a block-based language (Figure 1-3). Blocks are color-coded by their functions for easier navigation through the block dropdown menu (Figure 1-4). Each block has connecting corners that indicate which block pieces can fit together. Block code placed under an agent tab, for example “Turtle” tab in Figure 1-3, is applied to all agents of type “Turtle” while block code placed under “Everyone” tab is applied to all agent types created. Such agent-oriented model allows users to easily define behaviours of each agent and build complex simulations.

Figure 1-4 Dropdown menu for different block types. On the left, the outermost “while” block is an interface block that reads the state of an interface widget, “forever” toggle button. The blocks are color-coded for easier navigation. Similarly, “pen” block is an environment block that traces out the trajectory of an agent, while red “forward” and “left” blocks are movement blocks that instruct an agent how to move.
1.3 Educational Value of StarLogo

Thanks to color-coded blocks that can be put together like puzzle pieces, StarLogo Nova serves as a low-barrier entry point to programming and game development for elementary and middle school students. Students can learn computational thinking by playing with block pieces, instead of getting caught up in the syntactic details of a text-based programming language.

More importantly, StarLogo Nova can be incorporated into everyday classroom setting for various subjects outside computer science, where students can create and run large-scale 3D simulations of concepts such as diffusion of molecules, spread of epidemics, and ecology systems [6].

Since StarLogo Nova operates in a browser, it does not require special installation, and therefore, it is easy to deploy in a large classroom setting. In addition, StarLogo Nova supports project sharing and collaboration. As a result, teachers can share starting code with students and students can build on each other’s projects to create more complex models. These features make StarLogo Nova even more suitable for educational purposes.

1.4 Problem Definition

While StarLogo Nova has many applications in a classroom setting, there are limitations. Currently, StarLogo Nova does not support individual agent monitoring and mouse interactivity, tools that can enhance educational impact of StarLogo Nova even more. With agent monitoring, students can observe both overall phenomenon and behaviour of individual agents to link how a phenomenon is manifested in an individual and how an individual reflects the big phenomenon.
A complex system often consists of only a few simple rules. Yet it can be hard for students to break down an overall simulation by themselves to reveal a set of rules governing the phenomenon. In his book *Turtles, Termites, and Traffic Jams*, Mitch Resnick shares his experience of teaching gifted high school students how a pattern in the population can emerge from individuals responding to only local stimuli using simple rules. Even after weeks, many students believed there should be a central authority that coordinates the movement of the entire population to create a pattern or a complex system [2]. Ability to monitor any individual agent throughout the simulation can help overcome this gap. Agent monitoring maximizes the benefit of StarLogo Nova’s agent-based model and demonstrates how simple rules for individual agents are manifested in their behaviour in real time and how such behaviour builds up to patterns of a complex system over time.

Furthermore, ability to change a target agent’s properties through the agent monitoring pop-up allows students to provide direct inputs into a simulation, manipulate a simulation in an intuitive way, and trigger new events at the time and location of their choice. Such feature encourages dynamic user interaction with the simulation and highlights the connection between changes in an individual agent and its impact on the overall system.

There are workarounds inside the current version of StarLogo Nova to enable agent monitoring. Such workarounds, however, rely on manual code replication and synchronization, and therefore are cumbersome and prone to error. Moreover, unless users create workarounds for every single agent created throughout a simulation, they are limited to specific agents they created workarounds for and cannot interact with other agents in the simulation.
To enable efficient, targeted, and dynamic interactions between the user and StarLogo Nova simulation and broaden its impact, it is necessary to support agent monitoring and mouse interactivity.


2 Background

2.1 Common Use Cases

StarLogo Nova is free and runs in an Internet browser, with project sharing options. As a result, teachers can use simulations developed by the StarLogo community to illustrate complex concepts. Moreover, users can create simple and large-scale simulations using the library of agents or import their own custom sounds and 3D models. Consequently, different classes can customize the same simulation to fit topics taught in different subjects. Thanks to such benefits, StarLogo Nova has been introduced to students and teachers across the nation and has over 77,000 users [10].

In a classroom setting, StarLogo Nova users can create 3D games and simulations to study diverse concepts in science and math. Common topics include:

- Geometry
- Diffusion (Physics)
- Epidemiology
- Ecology

Following subsections describe each model and how agent monitoring can enhance users’ understanding of each topic.

2.1.1 Geometry Simulation

Users can learn about and explore geometric shapes as below:
1. A single Turtle agent is created.
2. The Turtle agent will repeat moving forward and turning by a user-specified angle, while “pen down” property traces the path the Turtle agent moves in.
3. The trace results in a regular polygon.

Users can change the size of regular polygons by changing the amount of steps forward, and change the type of polygon by changing the turn angle. Users can determine the turn angle by calculating either interior or exterior angle, which serves as mathematical exercise for students. Users can also extend beyond regular polygons and create other shapes such as a rectangle, a star, and a close approximation of a circle.

Since geometry simulation is the simplest simulation that involves only one agent, the application of agent monitoring is smaller compared to other use cases. Nevertheless, users can still observe the agent’s heading to see how the turn angles add up and observe changes in the agent’s x, y positions to see how it is affected by the degree of turn angle.

2.1.2 Diffusion Simulation

Users can simulate the diffusion of molecules as below:

1. Several Turtle agents are created. (The name of the agent type may be changed, for example to “Sugar”, to better illustrate the example.)
2. Starting x, y positions of Turtle agents are set close to each other so that the agents form a cluster.
3. Turtle agents move randomly at a user-specified speed.
Users can create more complex simulations by creating multiple types of molecule and customizing starting points and diffusion rate for each. Users can also handle collision for more realistic and interesting simulation.

At a first glance, users might think that there is a central rule that observes the overall simulation and directs individual agents to move towards area with lower density. Tracking the x, y positions of any agent through agent monitoring pop-up, however, reveals that each agent is moving randomly, unaffected by other agents. Observing the random movement of a single agent while witnessing the diffusion phenomenon at a higher level helps convey that a big, complex pattern is in fact caused by individuals following a simple rule, unaware of the pattern it is contributing to.

Furthermore, users can verify that changing the rules for individuals leads to changes in the individuals behaviour, and then the overall phenomenon. For example, if users introduce collision so that when individual agents collide they bounce off each other, users can observe irregular, abrupt changes in the agent monitor pop-up at the time of collision and alterations to the diffusion pattern.

2.1.3 Epidemiology Simulation

Users can create an epidemic model that simulates the spread of a disease through a population as below:

1. Multiple Turtle agents are created and programmed to move around randomly.

2. Turtle agents with a red color will have the disease, and healthy agents will be yellow.
3. Turtle agents will catch the disease by colliding with an agent who is already sick.

4. A sick individual will have a chance to recover at every step.

Users can customize variables for number of sick agents in the beginning, probability of catching the disease upon collision, and probability of recovering. In addition to simulation, StarLogo Nova supports real-time line graph that shows the number of sick agents and healthy agents over time (Figure 2-1).

![Figure 2-1 A line graph that shows change in population in healthy agents and sick agents over time. As the simulation runs, the graph displays both real-time updates and history since start of simulation.](image)
The line graph of the population displays a trend in the population and the relationship between the number of healthy agents and the number of sick agents. Complementary to the line graph, individual agent monitor pop-up shows that healthy agents and sick agents are not two separate groups but that the same agent alters between the two categories, contributing to trends observed in both. Users can find connection between the amount of time an individual agent becomes sick and the overall percentage of sick agents, and open multiple agent monitor pop-ups to compare how the same overall phenomenon is manifested differently in each agent.

2.1.4 Ecology Simulation

Users can create an ecology model that simulates the interaction between predator and prey as below:

1. Two types of agents are created and programmed to move around randomly.
2. Prey agents will be “eaten” and deleted by colliding with a Predator agent.
3. Both Predator and Prey agents can reproduce with user-specified probabilities.

Users can introduce further complexity by making a certain percentage of Predator “starve to death” when Predator-to-Prey ratio is too high, including probabilities of “natural death” for both Predator and Prey agents, and creating more agent types to account for more complicated food chain ecosystem. Similar to the case of epidemiology simulation, StarLogo Nova supports real-time line graph that tracks population of each agent type over time (Figure 2-2).
Figure 2-2 A line graph that shows change in population in prey and predator agents over time. The green line represents population of prey agent and red line represents population of predator agent [13].

While the line graph shows fluctuations in population, agent monitoring pop-up graphs lifespan of an individual. Users can open multiple agent monitor pop-ups to compare their life spans and how it influences the line graph of the population. Furthermore, this observation reminds users that similar principles apply not only to programmed models but also to them and real-life ecosystem, and that they too are contributing to the creation of a pattern without intending to do so. Users can also use agent monitor pop-ups to change x, y positions of the corresponding agents. By manually changing the environment, for example, aggregate prey agents away from predator agents, users can witness its impact on the overall phenomenon and verify that the overall complex system is determined by rules that define individual agents’ behaviour and the environment agents react to.
2.2 Motivation

While block-based language is simple and easy-to-use, the current version of StarLogo Nova does not support more complicated interactions between the user and agents. Mainly, user can observe and interact with agents only through pre-defined rules on an entire agent type. Types of events that agents can respond to are limited by functionalities provided by the block interface. Currently, StarLogo Nova detects a keyboard press and changes in interface widgets, but no mouse events. Supporting a mouse event is useful as it enables users to interact with individual agents and provide fine control over the simulation.

StarLogo Nova currently has a block type that graphs changes in an agent type’s properties as a group, but it is hard to track changes in an individual agent without creating a separate agent type or setting an arbitrary trait for that single agent. Even using such hacks, the target agent needs to be determined before running the simulation, and this constraint prevents dynamic interactions between the user and the simulation.

To broaden StarLogo Nova’s application in the classroom setting, it is useful and important to be able to monitor individual agents and perform dynamic, targeted interactions on them.
3 Agent Monitoring

3.1 Agent Monitoring in StarLogo TNG

StarLogo TNG, an older version of StarLogo Nova, had an agent monitoring feature [3]. When an agent was clicked in StarLogo TNG, a pop-up box with editable traits appeared (Figure 3-1). The user could manually edit each trait or graph changes in traits over time by clicking the graph buttons on the left.

![Figure 3-1 Agent monitor pop-up box in StarLogo TNG. Eight traits of the agent were displayed and could be edited by the user. Clicking the graph button on the left of each trait displayed changes in that trait over time.](image)

This feature has not been carried over to StarLogo Nova with the transition from downloadable programming environment to online programming environment. With the shift of platform, the
graphics rendering engine changed from OpenGL to Flash, and then later to WebGL. Agent monitoring in StarLogo TNG was implemented using OpenGL Utility (GLU) library, which is not supported by Flash or WebGL used to implement StarLogo Nova.

3.2 Current Workarounds for Agent Monitoring in StarLogo Nova

Currently, StarLogo Nova only supports overall trend visualization through line graph. The line graph depicts only properties of an entire agent breed, such as the number of agents. It is helpful, however, to be able to look into details of individual agents to truly understand the connection between individual agents and the overall phenomenon. For instance, clicking on a patient agent during an epidemic simulation and viewing changes in the health level of that patient over time visually conveys concrete details to students on how the overall epidemic phenomenon is expressed in individual agents. Observing a molecule agent’s movement during a diffusion simulation demonstrates that individual agent’s behaviour is the driving force that creates patterns in a complex system.

In the current StarLogo Nova programming environment, there are two workarounds to track individual agents.

3.2.1 Method 1: Create a separate agent type

The most intuitive workaround is separating out an agent by creating a new agent type. The user can create two agent types (e.g. “Turtle” and “SpecialTurtle”) with the same properties and behaviours. All agents except one will have the type “Turtle”, while the agent users wish to isolate will be the type “SpecialTurtle”. Since the code blocks are replicated to be the same, the
selected agent will behave exactly as other “Turtle” agents, while allowing users to monitor
behavior from an individual agent’s point of view by graphing properties of the
population-size-one “SpecialTurtle” breed.

3.2.2 Method 2: Set a special trait

An alternative workaround is keeping only one agent type and setting a special trait instead. In
the epidemic simulation, although both healthy agents and sick agents belong to breed “Turtle”,
sick agents are distinguished from healthy agents by setting color trait from yellow to red.
Special traits allow users to filter within the same breed and graph red “Turtle” agents and
yellow “Turtle” agents separately. Similarly, a single agent within the same agent type can have
a special trait (e.g. all “Turtle” agents have yellow color except one with red color), and users
can graph properties of that single agent by selecting to graph “Turtle agents with red color
trait”.

3.2.3 Limitations

The first method of creating a replica of each agent type is cumbersome and prone to error. Users
need to replicate all functions under “Turtle” agent tab to “SpecialTurtle” tab, as well as all other
functions outside “Turtle” agent tab that involve “Turtle” agent. Whenever one function changes,
users need to manually update the function for both “Turtle” agent and “SpecialTurtle” agent.
Such manual replication and maintenance is prone to error and hard to debug, and can easily lead
to inconsistent behaviours and inaccurate data.

The second method of utilizing a special trait is less cumbersome than the first method, but still
carries major limitations. For both the first and second method, if users wish to track multiple individual agents, they need to create corresponding number of special agent types or traits. One of StarLogo Nova’s strength is that it allows simulation with over 10,000 agents, and it is not practically feasible to set special agent types or traits one by one to sample even 1% of population (100+ agents) for large-scale simulations. More importantly, these workarounds enable monitoring only for predetermined individual agents with pre-set special agent type or trait. Unless users program to create a special agent type or trait for every single agent both at the start of a simulation and newly created throughout the simulation, users are restricted to a few agents they specified. Such constraints currently prevent individual agent monitoring and dynamic interactions between the user and StarLogo Nova simulation.
4 User Interface

4.1 Heuristics for User Interface

While there exist various heuristics and guidelines for user interface design, there are three main criteria consistently highlighted: learnability, efficiency, and safety [7, 11]. These three criteria are considered for the new design of agent monitoring pop-up.

4.1.1 Learnability

Learnability refers to how easily users can acquire knowledge about the interface using previous knowledge of the world. There are several factors that help improve learnability, such as consistency, affordance, and feedback [8]. Having consistent design within an application or with an external application or product allows users to quickly transfer their previous knowledge of the world by expecting similar behavior. Affordance is perceived and real properties of an object that determine how that object should be used. For example, arrows on the left and right sides of a webpage may indicate that users should either click on those buttons or press arrow keys to navigate through the page. On the other hand, a scrollbar may indicate that users should scroll their mouse or click and drag the scrollbar. Lastly, immediate feedback—especially visual feedback—informs users of consequences of their actions and whether their intention has been met or not.
4.1.2 Efficiency

Efficiency measures how quickly users can perform the task after they have learned how to do it. “Select All” button, keyboard shortcut commands, and autocomplete are examples of methods to improve efficiency. Slightly less intuitive examples include pre-filled default values and history in a web browser [8].

4.1.3 Safety

Safety has two main aspects: error prevention and error recovery [8, 12]. To prevent errors, dangerous commands such as delete should be placed far from common actions such as save, or be marked distinctively different. It is also helpful to provide additional confirmation dialogue for dangerous commands to make sure users intend to perform that action. In case users do make a mistake, the mistake should be reversible. “Undo” function in an editor and “cancel” button next to a progress bar are good examples of error recovery.

4.2 UI Design for Agent Monitoring Pop-up

When an agent is clicked in StarLogo TNG, a pop-up box with editable traits appears (Figure 3-1) [3]. Users can manually edit each trait or graph changes in traits over time by clicking the graph buttons on the left. While the agent monitoring pop-up in StarLogo TNG has good functionalities, there are UI improvements that can be made based on the three main criteria of usability: learnability, efficiency, and safety [8].

The old UI design in StarLogo TNG has high learnability. The graph button uses a graph icon for
simplicity. The up and down arrows next to property values clearly indicate traits to be editable. Therefore, the new design will follow the overall layout of the old design, with local changes. To increase efficiency, the new pop-up box in StarLogo Nova will display fewer traits that are more commonly used. It will focus on $x$, $y$, $z$ (height) locations, $size$, $color$, and $pen-down$ properties. Changes in $size$, $color$, and $pen-down$ make the selected agent easier to track with bare eyes. Changes in coordinates can be used to trigger new events, such as collision with other agents.

![Figure 4-1 Proposed new design for agent monitoring pop-up.](image)

Value fields will be simplified, and graph icons will be colored and simplified so that they stand out to users. *Invisible* property will be removed since it can lead to hard-to-recover errors and does not align with the purpose of agent monitoring.
The initial proposed design for agent monitoring pop-up is shown in Figure 4-1. *Invisible* property will be discarded to enhance safety. If users accidentally close agent monitoring pop-up while the monitored agent is *invisible*, it is hard to locate the agent and make it visible again. As a result, *invisible* property is no longer supported to provide protection against hard-to-recover errors.

The up and down arrows in each field will be eliminated as well. For x, y, *height*, and *heading*, altering the property value by one or two does not have visible effects. Therefore, arrows provide more distraction to users within the small pop-up window than being a useful control tool. Instead, users can simply type numeric values to update properties.

Lastly, the line graph icon will be replaced with a simpler icon for increased readability. The graph icon will be colored to distinguish it from black text, which will further help users compartmentalize different functions within the pop-up.
5 Technical Approach

Agent monitoring functionality consists of two parts: (1) detecting a mouse click event and recognizing which agent has been clicked, and (2) implementing a UI for displaying traits of the clicked agent.

The main challenge for agent monitoring UI lies in the design, not implementation, discussed in the previous section. For agent click detection, the problem is more complicated since the simulation represents a 3D world while the mouse click happens on a 2D monitor surface. A mouse click must be correctly mapped to the agent users intended to select. The next two subsections discusses two proposed solutions to the challenge of agent click detection: (a) ray casting and (b) color picking.

5.1 Ray Casting

The key idea of ray casting method is to create a ray from the camera to the position of mouse click and extend it through the scene. Then agents that intersect with the ray are reported in the order of intersection (Figure 5-1) [7]. The desired agent is the first agent the mouse ray (ray passing through the point of mouse click) intersects with.

Implementing click detection with ray casting, however, revealed the method to be not pragmatic given the current implementation of StarLogo Nova. StarLogo Nova can simulate up to tens of thousands of agents. To increase performance, StarLogo Nova uses instance rendering. Instead of creating and storing a ten thousand copies of the identical agent type, it only creates one
Figure 5-1 Diagram for ray casting. The left side of the square represents what is rendered on a 2D screen to the user. When a mouse click occurs, a ray is created from the camera position to mouse position on screen (ray entry point) and extended through the scene to find intersections. A mouse click at top red dot reports no intersection while a mouse click at the bottom red dot reports the blue sphere the ray intersects with [11].

While this technique helps reduce memory required and boost performance, it poses a challenge on using basic ray casting method. Since there is only a single instance of object created, ray casting only detects intersection with that instance. In other words, mouse ray reports no intersection with other rendered copies, even though those copies are visible on screen and are the ones the user intended to click.

This issue can be resolved by creating a new data structure that contains information on all copies, consistent with what is rendered on screen to users, and then performing ray casting on
this internal data structure. While it is possible to work around the instance rendering issue, color picking method is more efficient and accurate. As a result, agent click detection was implemented using an alternative method, color picking.

5.2 Color Picking

In color picking method, the program renders a separate scene off-screen, invisible to the user (Figure 5-2). In the off-screen framebuffer, each agent is uniformly colored with a unique color derived from its id. When the mouse is clicked, reading the pixel color value at the point clicked leads to recovery of corresponding agent id [1].

![Diagram for color picking](image)

Figure 5-2 Diagram for color picking. Scene on the left is rendered on-screen, visible to the user, and includes full shading. Scene on the right is rendered off-screen, invisible to the user, without any shading. In the off-screen framebuffer, each object is represented by a single unique color. Coordinates of a mouse click on canvas is transferred to the off-screen framebuffer, and the pixel value at the given position in the framebuffer identifies selected object [4].
Although rendering the scene twice may affect real-time performance, the color picking method has a major advantage: it works pixel-perfect regardless of different rendering modes used. StarLogo Nova has an alternative “2D rendering mode” for machines with low specs, where agents are pre-rendered in software and then rendered on screen as if copying over a 2D image. Ray casting would have yielded inaccurate results since agents are not spatially arranged in a 3D space. Color picking, on the other hand, only reads the final pixel value and the same algorithm can be applied in 2D rendering mode.

5.3 Implementing Color Picking

5.3.1 Intermediate Step for Debugging

Before focusing on the implementation of color picking, a small change was introduced for the ease of debugging. Usually, objects are rendered with a user-specified color or a random color if “random” option is chosen. To verify that the off-screen rendering is rendered correctly, rendering function was changed to assign color based on the agent object’s index \( i \). This modification assigns unique color for each agent object, and therefore developer can verify whether the intended agent has been properly selected through the pixel color.

```javascript
// ModelManager.js
for (var i = 1, len = agentStates.length; i < len; i++) {
    // ...
    this.batches[shape].colors[3 * j] = ((i & 0xFF0000) >> 16) / 255;
    this.batches[shape].colors[3 * j + 1] = ((i & 0x00FF00) >> 8) / 255;
    this.batches[shape].colors[3 * j + 2] = (i & 0x0000FF) / 255;
    // ...
}
```
The remaining code for color picking was first tested using on-screen rendering with the change introduced above, and then applied to off-screen rendering. For off-screen rendering, a new scene and texture was created to hold unique colors derived from agents’ ids.

```javascript
// Viewport.js
// Create scene and texture for off-screen rendering
self.pickingScene = new THREE.Scene();
self.pickingTexture = new THREE.WebGLRenderTarget(
  self.renderer.domElement.clientWidth,
  self.renderer.domElement.clientHeight
);
```

### 5.3.2 Agent Click Detection

To recognize and incorporate mouse interaction, a new mouse object and mouse click event listener were created.

```javascript
// Viewport.js
// Create mouse
self.mouse = new THREE.Vector2();
self.mouseClicked = false;

// Mouse click event listener
window.addEventListener( 'mousedown', function( event ) {
  self.mouse.x = event.clientX;
  self.mouse.y = event.clientY;
  self.mouseClicked = true;
}, false );
```

Mouse click event handler only updates mouse’s x, y location and boolean `mouseClicked`.

Afterwards, the renderer will continuously check the boolean `mouseClicked` to see if the view
needs to be updated.

// Viewport.js

// Function to update each frame
self.animate = function() {
  // ...
  self.needsUpdate = (self.needsUpdate || self.tweening) &&
  (now-self.lastUpdateTime > 33) || oneLastTween || self.mouseClicked;
  var proportion = self.tweening ? timeSinceLastExec / self.millisPerRun :
    1;
  self.render(proportion);
  // ...
}

render: function(proportion) {
  // ...
  if (this.needsUpdate || this.terrain.needsUpdate) {
    // ...
    this.renderer.render( this.scene, this.camera );
    
    if (this.mouseClicked) {
      this.renderer.render( this.pickingScene, this.camera,
                  this.pickingTexture );
      this.pick();
    } // ...
  } // ...
}

// ...

The next step is to read the pixel value at the location of mouse click. The pixel value is stored in pixelBuffer passed through the function, which can then be used to recover agent id.

// Viewport.js

pick: function() {

var gl = this.renderer.context;
var pixelBuffer = new Uint8Array(4);

// Read pixel value at the mouse's location
this.renderer.readRenderTargetPixels(this.pickingTexture,
  this.mouse.x, this.pickingTexture.height - this.mouse.y,
  1, 1, pixelBuffer);

// Recover agentID
var agentID = ( pixelBuffer[0] << 16 ) | (
  pixelBuffer[1] << 8 ) | ( pixelBuffer[2] );

// ...
}

5.4 Implementing Agent Monitoring Pop-up

A pop-up is displayed when an agent has been clicked and the agent does not already have a pop-up open. Pop-ups are implemented using jQuery’s dialog widget and a table inside the dialog widget to align properties [5].

Each dialog widget is assigned a different HTML element id to enable users to have multiple pop-ups open for multiple agents and track and update them independently.

// Viewport.js

pick: function() {
  // ...
  var agentState = this.agentStates[agentID];
  // ...
  var traits = '<tr><td><img id=agentID+x alt="" title="" src=img/graph-icon.png'/></td>' + '<td><td><td>' + agentState["x"] += '</td></tr>'+
    '<tr><td><img id=agentID+y alt="" title="" src=img/graph-icon.png'/></td>' + '<td><td>' + agentState["y"] += '</td></tr>''}
All property values in an open agent monitoring pop-up are updated when there is a change in view and renderer is updated. Below code snippet shows an example of updating monitored agents’ x values, which can be looped over and repeated for other properties.

The only exception to updating property values in real-time is when the value textbox is selected by users. This indicates that users may be trying to edit the property. In this case, updating the property value would refresh the field and prevent users from inputting a new value, given the high rendering speed.

```javascript
// Viewport.js
render: function(proportion) {
    // ...
}
```
User update is captured and handled by the following code. The code takes action on `blur`, or when user clicks outside the property textbox to exit out of it. Upon `blur`, the agent property is updated with new user-provided input.

```javascript
// Viewport.js

pick: function() {
    // ...
    if (!(agentState == null)) {
        var popup = document.getElementById(dialogID);
        popup.addEventListener('blur', function( event ) {
            var userVal = event.target.value;
            var targetTable = event.target.parentElement.parentElement;
            var targetAgentID = targetTable.rows[0].cells[1].innerHTML;
            // ...
        });
    }
    // ...
}
```

By clicking on the graph icon next to each property, users can view a line graph to start viewing change in the property in real time. A code snippet for attaching event listener to one trait for one agent is shown below. This code snippet is repeated in for loop for each agent and each trait.

```javascript
// Viewport.js

pick: function() {
  // ...
  if (!!agentState == null) {
    // ...
    document.getElementById(targetID).addEventListener("click",
    function(){
      WidgetManager = slnova.WidgetSpace.getInstance();
      WidgetManager.addWidget("Agent Monitor", "ChartWidget", true);
      lineGraph = WidgetManager.widgets_[
        WidgetManager.widgets_.length - 1];
      lineGraph.setXAxis("clock");
      lineGraph.setYAxis(targetTrait);
      lineGraph.addData("Series 1", Common.State.clock,
        agentState[targetTrait]);
      lineGraph.redraw();
    });
    // ...
  }
  // ...
}
```
5.5 Supported Features

Through the current version of agent monitoring feature, users can click on any agent to display a pop-up that displays $x$, $y$, $z$ positions, heading, size, color, and pen-down. Each field accepts manual input from the user, which is immediately reflected in the simulation when user exits the textbox by clicking outside of it. The user-made changes stay even when users close the corresponding pop-up of that agent. Users can monitor multiple agents simultaneously through multiple pop-ups, as well as move around pop-ups to organize them. Users can also click on a graph icon to view a line graph that shows changes in the selected property over time.
6 Design for Extending Mouse Interactivity

Agent monitoring through mouse click detection is an exciting feature on its own, but it has even greater implications. With the ability to detect which agent has been clicked, this functionality can be extended to other types of mouse interactivity. Instead of automatically displaying an agent monitoring pop-up, StarLogo Nova can incorporate mouse interactivity to enable users to execute custom block code when a mouse event occurs, where a mouse event extends beyond a mouse click to a double click and a drag. Mouse events can be used as a dynamic trigger event that users can control more easily. This section outlines the design for extending mouse interactivity that may be used by future developers.

6.1 Significant Use Cases

6.1.1 Custom Click, Double Click, and Drag Events

Incorporating mouse interactivity allows for much more interesting and complex user interactions with the simulation. By selecting a specific agent through a mouse click, users can trigger a new event. For instance, a mouse click on a cell agent can be used to initiate a mutation in the clicked cell. Users can also click on the terrain. In agent monitoring, if a pixel color detected matches the terrain color, the click is ignored. This could be easily changed to enable mouse click events on the terrain. For example, users can click on the terrain to create a new agent at the location of the click. In this case, the agent will be created at height 0, right on top of the terrain, unless otherwise specified in the user block code.
A double click behaves similarly to a click, but allows for a different event trigger. In a way, detecting a click and a double click separately is similar to having two handlers for a click. For example, a double click can launch the agent monitoring pop-up while a click turns a healthy agent sick. Alternatively, a click and a double click can initiate two different types of diseases.

A drag event enables different types of interactions from a simple click. Users can drag an agent to reposition it, to manually update an agent’s position without opening an agent monitor pop-up and changing x, y values separately. In a physics simulation, a mouse drag event can represent an external force acting upon an agent and influence the simulation.

A mouse hover event is not handled as it is too expensive to continuously check for agent selection and distracting for users, especially in a big simulation. Instead, only events with clear user intention, such as a click and a drag, are handled.

### 6.1.2 Game Control

So far, this thesis focused on 3D simulations instead of game development because a simulation has more direct, relevant educational applications. Moreover, agent monitoring is not as useful or necessary in a game since important system variables, such as score or lives, tend to be displayed explicitly. Extended mouse interactivity, however, has interesting and important implications on both a simulation and a game.

In addition to new functionalities mouse interactivity introduces, mouse interactivity is important in terms of user experience. For most users, mouse control is easier and smoother than keyboard control. Especially in a game development, adding support for mouse control allows for faster
and more targeted user interactions than what is currently provided by keyboard control. For example, to move to the end of a path in an adventure game, user simply needs to click the target location instead of continuously pressing an arrow key. In a bubble shooter game, changing the aim using keyboard requires pressing an arrow key multiple times until the desired angle is reached. On the contrary, with mouse, user can simply click the final target location and the aim is updated immediately, skipping unnecessary intermediate steps.

**Figure 6-1 Different effects of left and right keyboard navigation on bird’s-eye view and first-person point of view.** In bird’s-eye view, left and right arrow keys lead to moving left and right down the path. To the contrary, in first-person point of view, left and right arrow keys lead to turning left and right by 90 degrees, whereas up and down arrow keys move the character forward and backward along the path.
Furthermore, mouse control provides more intuitive interactivity than keyboard, especially when switching between a bird’s eye view and a first-person point of view. In bird’s-eye view, if there is a horizontal path with a character in the middle, pressing the right arrow key results in movement to the right (Figure 6-1). In first-person point of view, however, pressing the right arrow leads to the character turning to the right by 90 degrees [14]. To move the character along the path, the user must press up and down arrow keys instead. The correlation between keyboard control and its effect is not as consistent as mouse control, which maps user’s intention directly to target mouse coordinates.

6.2 Design of Mouse Blocks

To allow users to define their own response to a mouse event, new mouse blocks need to be created. To handle mouse click, double click, and drag events while preserving support for agent monitoring feature, blocks should be added to three drawers: “Detection,” “Traits,” and new “Mouse” drawer.

![Figure 6-2 New blocks in “Detection” drawer.](image)

**Figure 6-2 New blocks in “Detection” drawer.** Users can customize what will happen on a click, double click, and drag event through these new “Detection” blocks.
In “Detection” drawer, blocks of click, double click, and drag event handlers need to be added (Figure 6-2). Through these blocks, users can specify what action to take when a mouse event occurs. “On click” and “on double click” blocks can be placed either in a specific agent tab or in the “world” tab, whereas “on drag” block can only be placed in a specific agent tab. “On click” and “on double click” blocks placed in a specific agent tab will define what happens when an agent of that type is clicked, while those placed in the “world” tab specifies what happens when the terrain is clicked. The “on drag” block will only be used in a specific agent tab since it implies movement. This may be changed later, however, if new functionalities that imply movement of the terrain are added. For example, if 3D terrain edit is enabled so that a terrain can have bumps instead of being a single flat surface, a mouse drag event on the terrain can be used to change the height of the selected point on the terrain to create custom 3D terrain shapes.

![show my agent monitor popup](image)

**Figure 6-3 A new block in “Traits” drawer.** This new “Traits” block preserves agent monitoring feature.

To provide continuous support for agent monitoring feature, an agent monitor block is added in “Traits” drawer (Figure 6-3). Combined with “on click” or “on double click” block, this block
launches the agent monitor pop-up of the clicked agent. It is possible to use the block with “on
drag” block as well, although it is not a commonly-expected use case. Agent monitoring is not
applicable to terrain, and therefore can only be used in agent tabs, not “world” tab.

Figure 6-4 New boolean blocks in “Mouse” drawer with dropdown menu. A semicircle
connector indicates that the block is a boolean. These blocks are booleans for mouse events that
can be used with “Logic” blocks such as “if.”

Lastly, a new “Mouse” drawer that contains variables for mouse events needs to be created. The
first type of “Mouse” blocks are booleans (Figure 6-4). The boolean blocks indicate whether a
mouse event happened, and can be used with “Logic” blocks such as “if” or to create other
boolean values. For click and double click events, which can be used in either agents or “world”
tab, users can choose between all click events, click events of only agents, or click events of only
the terrain. A drag event is limited to agents tab, and therefore the dropdown menu has only
“agent” option.
Figure 6-5 New numeric blocks in “Mouse” drawer with dropdown menu. A square connector indicates that a block has a numeric value. Users can access numeric values of $x, y$ coordinates of the position of a click, as well as star and end $x, y$ coordinates of a mouse drag.

The second type of “Mouse” blocks are numeric (Figure 6-5). Through these blocks, users can get $x, y$ coordinates of the position of a mouse click or a double click. For a mouse drag, users can access $x, y$ coordinates of both the starting position and the ending position. Users can use these values to update positions or perform related calculations.

6.3 UI Design Principles for Mouse Blocks

New blocks added for mouse interactivity have three separate blocks for click, double click, and drag events, instead of having one block with a dropdown menu with three options. This design was inspired by the design of current “Keyboard” drawer, which has two separate blocks for two keyboard actions: typing and holding (Figure 6-6).
Figure 6-6 Keyboard blocks in StarLogo Nova. There are two separate blocks for two keyboard actions: pressing a key quickly ("typed") and holding it for a long time ("held").

Having separate blocks instead of one block with a dropdown menu of all possible actions seems less efficient. This design, however, increases learnability and safety, the two other important aspects of usability [8]. Since mouse blocks are designed based on existing keyboard blocks, users can transfer their current knowledge of how to use keyboard blocks to new mouse blocks, which increases learnability. More importantly, having separate blocks reduces the chance of choosing a wrong mouse event and introducing unintended behaviors and hidden bugs. This increase in learnability and safety outweighs decrease in efficiency.
7 Future Work

While color picking method for agent click detection works well, there are still some challenges. First, since StarLogo Nova uses a 24-bit color, when more than $2^{24}$ (approximately 17 million) agents are created, the RGB color value generated from the agent id might be outside the range of supported pixel values. Second, zoomed agent images can result in blurry boundaries, which may produce different pixel values within the same agent. These edge cases were not made priority of this thesis since it is unlikely for users to create over 17 million agents or choose to click on the border of a large agent model image instead of the center. In later versions, however, these edge cases should be addressed for precise, pixel-perfect agent click detection.

In addition, the design of extended mouse interactivity discussed in the previous section needs to be implemented.
8 Conclusion

StarLogo Nova is an online programming environment that utilizes easy-to-understand block programming language to allow any users to create a 3D game or simulation. It is widely used in elementary and middle schools to demonstrate STEM concepts such as geometry, diffusion of molecules, epidemiology, and ecology.

As powerful of a tool StarLogo Nova is, one of its drawbacks is limited support for user interaction with individual agents in the simulation. Current version of StarLogo Nova only supports overall monitoring of the entire agent type, and users must find cumbersome workarounds to be able to inspect details of a single agent. Moreover, such workarounds are not scalable to large simulations.

With the implementation of agent monitoring, now users can click on any agent to inspect changes in its properties in real time, as well as input values to update an agent’s $x$, $y$, $z$ positions, heading, size, color, and pen-down properties. Multiple agents can be monitored simultaneously, and pop-ups can be moved around and laid out as users wish. Clicking on a graph icon next to a property displays a line graph that tracks changes in the selected property over time. Agent monitoring provides users with an inspection tool to gain concrete understanding of individual agents’ behaviour and connect it with the overall trend of the population.

Moreover, the principles of agent click detection and agent monitoring can be extended to broader mouse interactivity, which can provide even more dynamic and granular user
interactions. Mouse interactivity will be a feature with high learnability due its structure similar to existing blocks, yet it will opens up a whole new domain of applications and use cases on the users’ end.
References


http://api.jqueryui.com/dialog/.


http://education.mit.edu/portfolio_page/starlogonova/.


