Data Analytics and Visualizations for StarLogo Nova Block Programming Platform

by

Phoebe Hiuchin Tse

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

September 2017

Author .....
Department of Electrical Engineering and Computer Science
July 24, 2017

Certified by .....
Eric Klopfer
Director of MIT Scheller Teacher Education Program
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Data Analytics and Visualizations for StarLogo Nova Block Programming Platform

by

Phoebe Hiuchin Tse

Submitted to the Department of Electrical Engineering and Computer Science
on July 24, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented an analytics tool for StarLogo Nova, a block-based programming platform for creating 3D simulations and games. This tool collects block data from the platform and visualizes the data into 15 different charts, showcasing both high-level and the more detailed aspects about the data. As a way to investigate how to improve StarLogo such that it is more accessible to beginner users and more extensible to experienced users, the tool includes charts that focus on the following four elements: tutorials, breed tabs, scrolling, and block size and placement. Suggestions for how to improve StarLogo's user interface, driven by a few runs of this tool on the data, are documented in this thesis as well.

Thesis Supervisor: Eric Klopfer
Title: Director of MIT Scheller Teacher Education Program

Acknowledgments

I thank the members of the Scheller Teacher Education Program (STEP) lab for their support, encouragement, and insightful feedback on my work. I specifically thank my direct supervisor Daniel Wendel, who was always available to discuss everything from the vision to project pivots to specific technical issues throughout the entire process to thesis revisions. Also, I would like to thank Eric Klopfer for being willing to revise and revise, as well as Paul Medlock-Walton for encouraging me to think beyond StarLogo toward the vision of improving the block-based programming space as a whole. I hope that the tool developed and insights discovered in this thesis project contributes to the future of StarLogo as well as the broader space of block-based programming.

Contents

List of Figures

Chapter 1

Introduction

StarLogo Nova is an online agent-based block programming environment that allows users of a wide age range to build 3D simulations and games, in order to visualize and better understand complex systems. By the end of Summer 2016, the engine running the entirety of StarLogo Nova was completely rewritten from ActionScript (Flash) to JavaScript [21]. Because the system underwent a huge overhaul, we decided to take a step back and focus on how to make the StarLogo Nova platform better: easier for beginners and more extensible for advanced users. In this chapter, we give a brief description of StarLogo Nova and its users. We also discuss the specific areas of focus in our investigation and give an overview of the rest of this thesis.

1.1 StarLogo Nova Platform Overview

StarLogo Nova is a web-based platform that allows users to create 3D models, games, and simulations using programming blocks instead of textual code. The platform teaches users how to code with blocks, so anyone, not just those who already know how to write textual code, can build with the platform. StarLogo is agent-based, which is useful in modeling how individuals or organizations interact with and are affected by their environment and the system as a whole. Thus, one can use StarLogo to create educational applications in many fields, including everything from ecosystems to epidemiology.

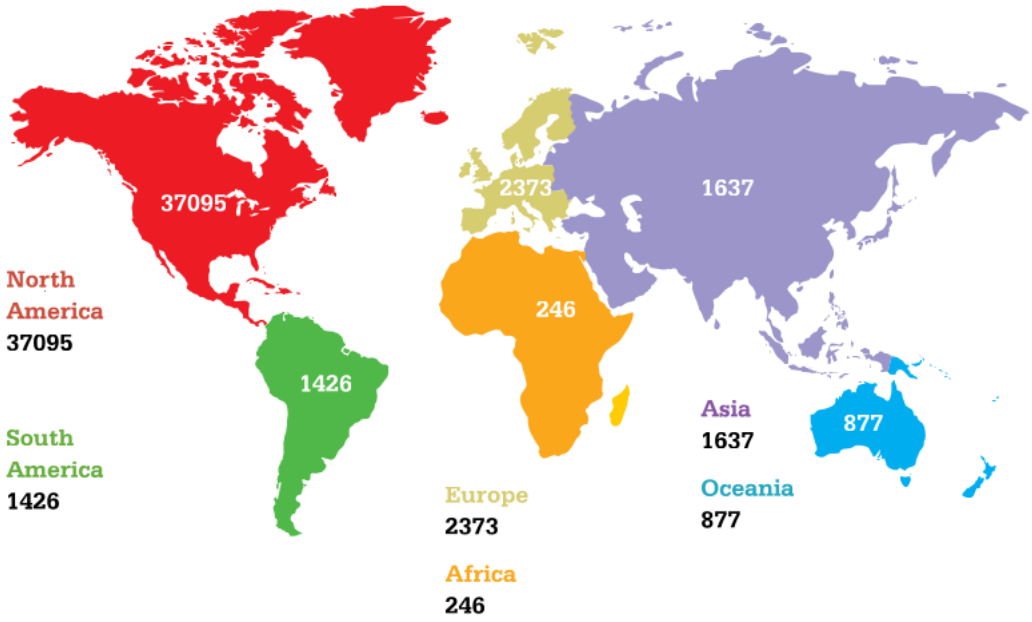


Figure 1-1: Teachers With GUTS User Map

1.1.1 Introductory Programs

Project GUTS

Project Growing Up Thinking Scientifically (GUTS) is a major source of users for StarLogo. Beginning in 2007 and initially held for middle and high school students in New Mexico, this program introduces computational thinking within math and science concepts, specifically abstraction, automation, and analysis [18] [8]. With the Use-Modify-Create model, teachers use MIT's StarLogo to teach students how to create 3D models and simulations of ecosystems and other complex systems [17]. This typically means that students are given pre-built StarLogo projects to use, then given tasks that require modifying the projects, and then encouraged to make their own models on the platform [19].

Since its beginnings, GUTS has grown, partnering with code.org to offer GUTS' curricula, and therefore StarLogo, to schools across the nation [3]. 43,654 individuals across the globe have visited the GUTS site or registered for a Project GUTS online course, as seen in Figure 1-1 [14]. Within recent years, Teachers with GUTS (TWIG), an online and in-person professional development network of teachers in the Project

GUTS community, has grown across the globe as well [14]. Within TWIG professional development courses, teachers are taught different ways of incorporating StarLogo models into their classroom curricula and how to use StarLogo themselves.

Examples of introductory projects found in Project GUTS include Flower Turtles, which is a basic project in which turtles draw flowers on the terrain, and the Epidemic Model, which simulates a disease's rate of infection in a population. Many students and teachers who have participated in any part of Project GUTS and/or TWIG have probably been exposed to projects like Flower Turtle and the Epidemic Model before.

Imagination Toolbox

Another source of users is Imagination Toolbox, a program held by the MIT Scheller Teacher Education Program (STEP) Lab every year [7]. During this annual week-long workshop, 20-30 teachers come together to learn how to build 3D games and simulations in StarLogo, and how to incorporate the platform into their classrooms [7]. Both formal and informal educators of any field are welcome to attend. StarLogo users are directly attained through these annual workshops, and indirectly through these teachers' students outside the workshop.

Introduction to Programming with Imagination Toolbox (IPWIT) is a series of tutorial projects that Imagination Toolbox workshop attendees are guided through. This series of projects includes Orientation, which is essentially the Flower Turtles project mentioned in the previous section, as well as other projects like a Paintball shooting game or a Treasure Hunt game [6]. There is overlap between the projects introduced in Project GUTS and those introduced in IPWIT. Additionally, these IPWIT tutorial projects are available publicly on StarLogo's resources page, so any user using StarLogo can use these starter projects to learn how to use the platform.

1.1.2 Users

How Beginner Users Learn StarLogo Nova

As discussed in the previous section, a majority of end users are introduced to StarLogo Nova through some structured program with beginner tutorials. From our understanding of how most users are introduced to the platform, we know that users range from elementary school students to teachers and researchers. They often learn from these tutorials, which include a series of guided instructions on how to make 4-7 types of 3D models and game projects. These guides contain screenshots and detailed instructions on how to create 3D simulations by connecting the block pieces together [6]. The tutorials in IPWIT also include detailed screenshots on how to use the website itself, such as how to create a project and how to navigate between different pages [6]. They also define the various terminology commonly used in the system, such as 'Agent' and 'Breed of Agents'.

It is therefore expected that most beginner users, no matter the age, have projects that look similar to those provided in the tutorials. Their projects might not be very large and could look like direct copies of existing tutorial projects. We do not know exactly how users' projects differ from the starter projects introduced to them in programs like GUTS or IPWIT, though.

How Advanced Users Use StarLogo Nova

Once a user grows more familiar with the StarLogo platform, they may begin to include more advanced, complex logic into their programs, straying further from tutorial projects. Just as complex text-based coding programs can get disorganized and more difficult to read, StarLogo Nova's complex projects can explode in size. For example, in Figure 1-2a, we see a partial screenshot of a very tall stack of StarLogo blocks. The user is interested in creating many agents and modeling relatively complex behavior. As such, the stack of blocks is very tall. Additionally, because of the nature of StarLogo blocks, the blocks' widths can grow very wide very quickly. This is seen in Figure 1-2b, which is a partial screenshot of multiple levels of nested blocks

[11].

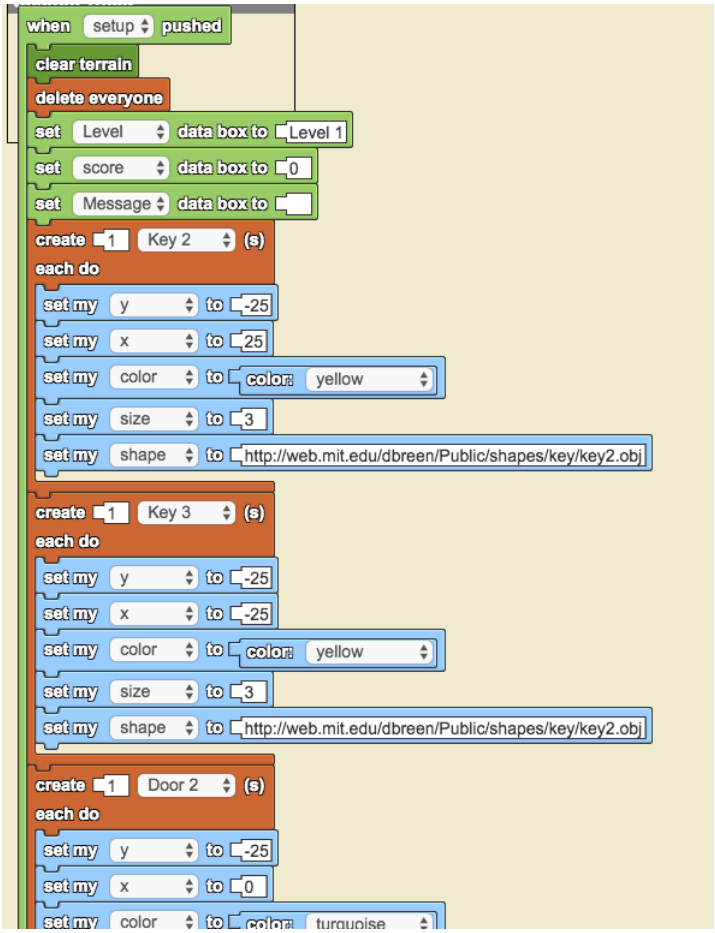
1.1.3 Interface

The development space on the website is, for our purposes, the page of greatest interest. On this page, end users create their projects with drag-and-drop block code, run their code, and see their 3D models and games come to life. Figure 1-3 displays what the development page looks like in edit mode [11]. In this mode, the page is divided into 3 sections: the Project Information section at the top, Spaceland in the middle, and the Workspace (Canvas and Drawer) at the bottom. The Canvas is where blocks are placed, and the Drawer is where users can find and select the different types of blocks that they want to use. Draggable and droppable, these blocks can be connected together like puzzle pieces to create 3D games and simulations. In play mode, the page hides the Workspace section such that blocks are not visible to the end user. To navigate between any of these three sections, the end user is required to scroll up and down the development page.

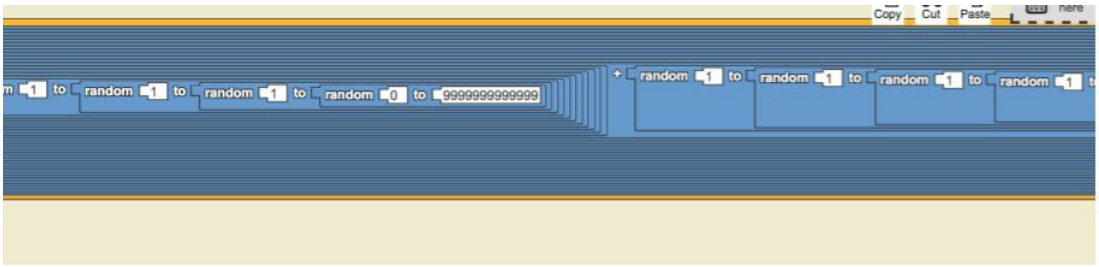
Currently, there are a total of 108 different types of blocks that end users can use in their programs. The block code can be run by pressing the *RunCode* button at the top of the page. The resulting program is then visible in Spaceland, where end users can see their 3D program's button interfaces as well as their agents interacting with one another.

1.2 Main Elements for Investigation

From my own interactions with the StarLogo Nova interface and from discussions with StarLogo researchers, we established four initial areas of investigation that may be useful for improving the platform's accessibility to both new and expert users. We used these four factors as the first step in thinking about what we wanted to learn about how users use the platform, and ultimately how to make the platform easier for beginners and more extensible for experts. The following sections describe these four elements and pose questions of interest regarding each.



(a) Tall Stack



(b) Wide Stack

Figure 1-2: Example of Tall and Wide Stacks

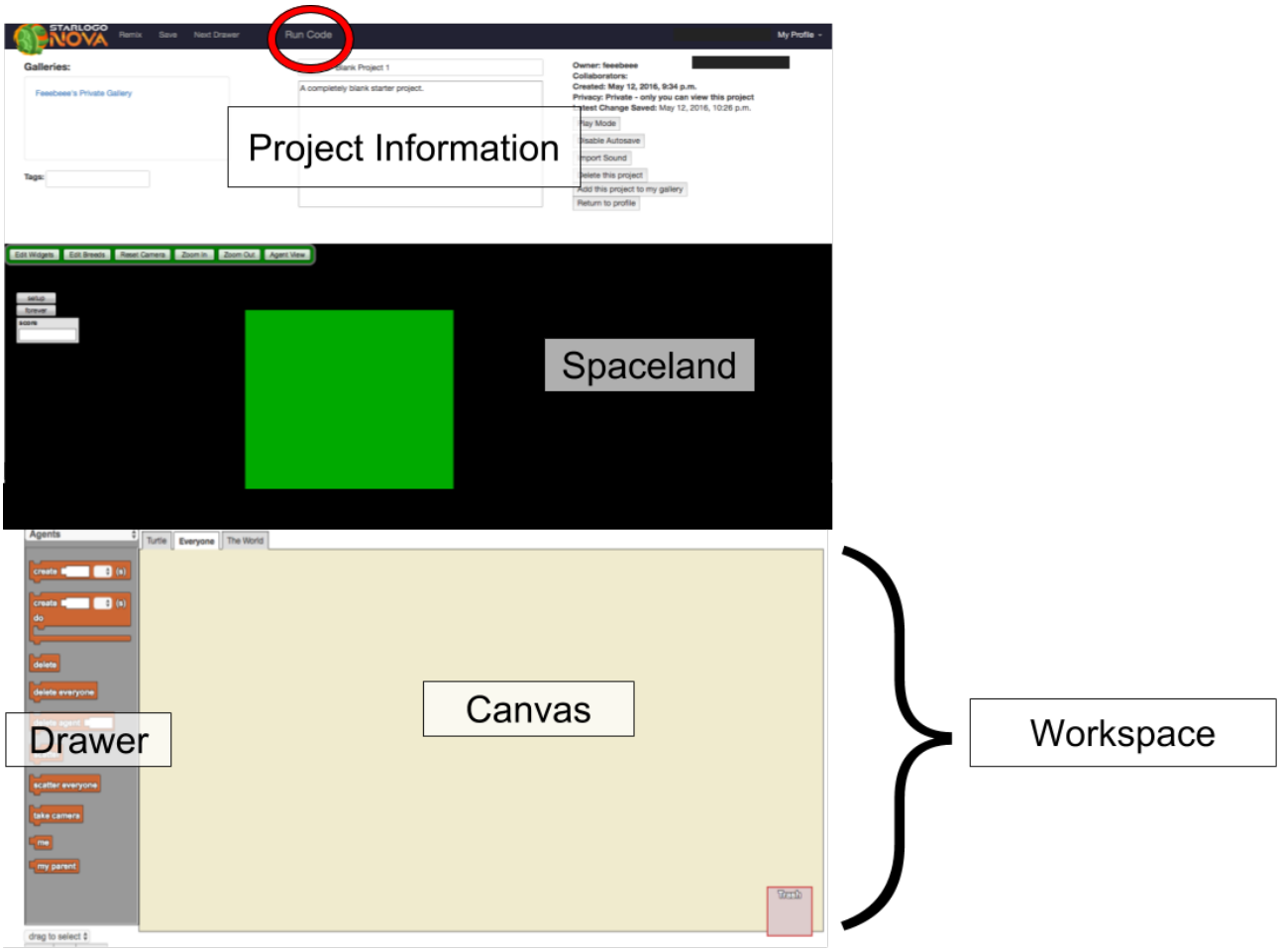


Figure 1-3: StarLogo Development Page

1.2.1 Tutorials

In order to fully and thoroughly learn StarLogo Nova, one must be able to read (and have the patience to read) the large set of instructions in the IPWIT tutorial guide, which was discussed in Section 1.1.1, Introductory Programs. Aside from written instructions, some additional in-person guidance is generally needed to explain the guide and explain how to use the website. While this may reveal a need for an improved orientation process to StarLogo Nova, this could potentially reveal a lack of intuitiveness in the interface as well.

According to StarLogo Nova developers, over 90% of the projects on the platform are expected to be from the last three years, mostly because of Project GUTS [8]. However, we do not have documentation on how the platform was introduced to those who have not directly participated in these programs. It is unclear whether teachers participating in GUTS or Imagination Toolbox had ever incorporated what they learned about StarLogo in their curricula. If they did, it is still unclear exactly how they introduced the platform to students.

Thinking about tutorials and their effect on users raises important questions for evaluating the platforms' accessibility to new users. How many of the projects are duplicates of, or very similar to, tutorials? Exactly how different from the tutorials are the more complex projects? Do users take the platform and expand upon it? Answering these questions would bring interesting insight into how users use the platform and how effective StarLogo Nova is in facilitating creative innovation.

1.2.2 Breeds

Agent-based programming allows for more decentralized thinking and thus allows for modeling of emergent phenomena [20]. An agent has its own pre-defined set of traits and behavior, and cannot change other agents. Instead, agents can interact with other agents and react from said interactions. This type of programming is particularly useful for modeling complex systems such as ecosystems in nature.

A breed is defined as a 'prototype for an agent' and is a term used particularly in

StarLogo [4]. This means that all agents of the same breed have the behavior defined by the block code given to that breed. This 'breed' concept is a way to help end users better understand how to code different types of agents and how to better organize their StarLogo projects. On the platform, there are two special breeds called 'The World' and 'Everyone'. 'The World' is a special breed that is the only one in which other agents can change its traits. Also, 'The World' is the only breed that exists when a StarLogo game or simulation is loaded, so it is naturally the breed where most agent creation occurs.

The 'Everyone' breed contains the blocks that define any shared behavior between agents. If the 'Everyone' breed tab did not exist, any project in which agents possessed shared behavior would have the shared behavior blocks repeated for each breed tab. Having 'Everyone' was a way to help cut down on repetitive code. When breeds do share behavior, they generally do so using Procedure blocks. For example, in a simplified ecosystem model, both a Lion agent and an Antelope agent may grow in size after eating. The Lion would grow larger after eating an Antelope, and the Antelope would grow larger after eating grass. The shared behavior here is 'growing in size', but they occur in different situations. Wrapping the blocks that define 'growing in size' in a Procedure block in 'Everyone' allows for both Lion and Antelope breed blocks to call the Procedure when defining individual 'eating' scenarios, without repeating code.

When users first begin creating a project, they are shown three default tabs that represent three breeds: 'The World', 'Everyone', and 'Turtle' (a basic breed). These tabs are similar to web browser tabs. When a tab is selected, the canvas that represents that particular breeds' blocks is displayed. The 'Everyone' and 'The World' tabs look the same as any other breed in the project, despite being conceptually special.

Similar to how one can view one tab at a time in a standard web browser, StarLogo users can only view one breed at a time and therefore do not have the ability to visually compare blocks between breeds at the same time. In addition, blocks can be copied and pasted across breeds, but it is accomplished by a series of mouse clicks, and not the usual drag and drop motion usually associated with these blocks.

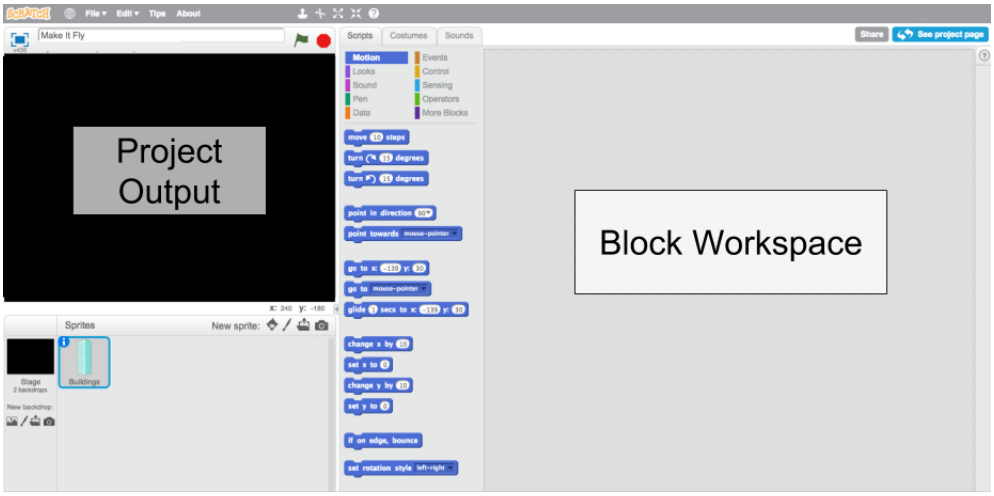
Is the breed tabs layout getting in the way of users' interaction with the platform? Are users using the 'Everyone' and 'The World' breeds the way researchers had anticipated? Are users' blocks actually organized because of these different tabs of breeds? Researchers are wondering about these questions and about how breed tabs affect both new and expert users' experiences with the platform.

1.2.3 Scrolling

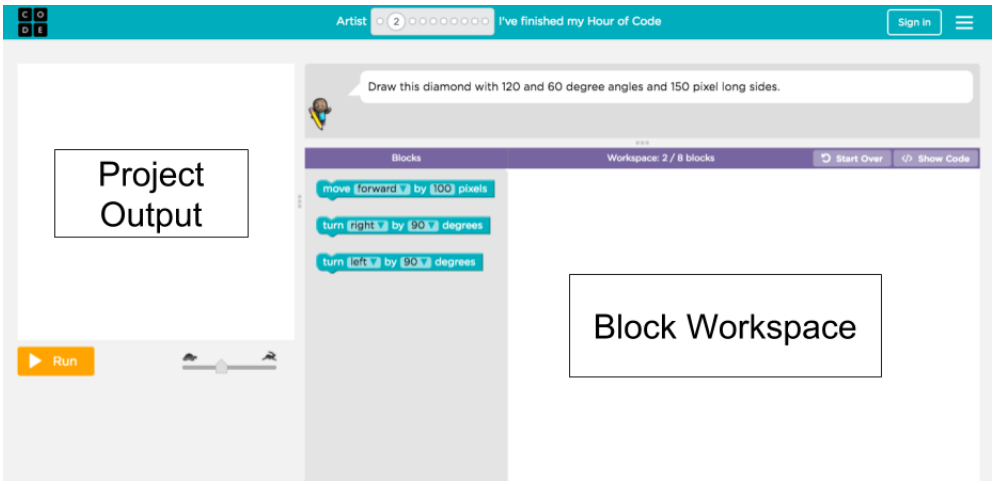
As seen on Figure 1-2, the Workspace section, which contains the Drawer and the Canvas, is below Spaceland. While working on their blocks, end users are not able to see the running environment next to the block environment. Users cannot concurrently view Spaceland and their blocks. Being forced to only view one environment at a time can make it difficult for end users to debug their projects. The platform's required scrolling could interfere with a new user's ability to explore the StarLogo Nova system to its greatest capacity, depending on the amount of scrolling. Requiring users to scroll up and down between their blocks and their program generated by the blocks is different from many of the popular block-based programming platforms that currently exist- such as Scratch, code.org studios, and App Inventor [9] [3] [2] . For Scratch and code.org, the canvas of blocks is on the right and the output window is on the left. Scratch (Figure 1-4a) and code.org (Figure 1-4b) have similar layouts. For App Inventor, the project output and the block workspace are on separate pages. To navigate between them, users must click on buttons located in the top right corner. This is pictured in Figure 1-5.

Overall, these other platforms' layouts do not require as much vertical scrolling for the user as StarLogo does. Also, StarLogo researchers have noted that users are oftentimes required to scroll horizontally because when complexity increases, the blocks tend to grow wider. The visual effect of such expansion is pictured in Figure 1-2b.

How often do users need to scroll up and down when developing? How often do they need to scroll left and right? Does this amount of scrolling impact usability? Would changing the layout of the development page improve usability and learnability



(a) Scratch Layout [9]



(b) Code.Org Layout [3]

Figure 1-4: Scratch and Code.Org Development Pages

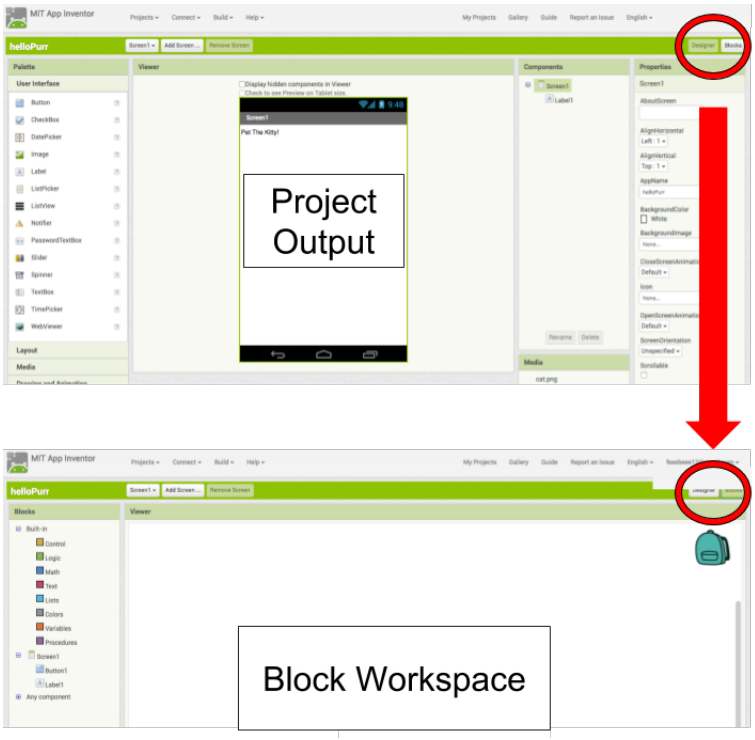


Figure 1-5: App Inventor Development Pages

of the system? These are questions that StarLogo researchers have not looked into yet, but are interested in answering. Gathering data on how screen real estate is used could inform researchers how to redesign the development page such that scrolling is minimized, and thus, hopefully improving the platform for new and advanced users.

1.2.4 Block Size and Placement

Blocks can explode in width incredibly quickly. This is especially common if the user wants to use math blocks to compute certain values, since math expressions can have many nested arguments, which could mean potentially many nested blocks. Stacks of blocks can also grow very tall as projects grow in complexity. As users explore the StarLogo platform, and as they gain more expertise with the system, the blocks may become more and more difficult to organize and understand.

Aside from the actual size of each block, the placement of blocks and stacks of blocks can also make one's project visually confusing. For example, unused blocks and unused stacks of blocks can be scattered all along the canvas without affecting

the program. With this potential visual overload of both active and inactive blocks, users may find it difficult to understand how their own programs and blocks work.

Where are users' blocks usually placed on the canvas? How large or small are these blocks usually? How many projects in the database have abnormally huge block stacks? Answering these questions would give insight into how visually complex projects are, and better inform any future user interface design decisions.

1.3 Thesis Overview

What does a project's similarity to or difference from a tutorial starter project say about StarLogo's ability to encourage creativity? Does the current layout help or hinder users' experiences with the site? Does the platform's enforced need to scroll frustrate users? Do the blocks' sizes and positions make the platform visually overwhelming? Without analyzing data on how users use the platform, these questions will remain unanswered. With a greater understanding of the data, though, we can begin to address these questions.

With data, we can ask more specific questions that would bring us closer to understanding StarLogo's specific impacts on users. How do StarLogo projects look in the database? How similar to and how different from tutorials are projects? Are breed tabs organizing users' code effectively? What is the extent of required scrolling on the development page?

This second set of questions, which can be answered with data analysis, motivates this M.Eng project. Answers to these questions would equip us with the necessary tools to begin evaluating StarLogo's ability to encourage creativity and to determine what design changes would improve user experience. This would thus give insight on how to improve the platform for both beginner and advanced users.

The overall goal of this thesis is to create an analytics tool that collects data about the projects on the StarLogo Nova platform and visualizes said data in an informative matter. The tool is aimed to be effective in displaying relevant data that answers our questions and is easy to build upon in the future. The analysis that is presented in

this thesis is an overall look at how users interact with the StarLogo Nova platform. At the end of this thesis, I provide several recommendations for the redesign of the StarLogo development page, informed by the data visualization results gathered from our data analytics tool.

This chapter described the background and initial directions for StarLogo's new data analytics and visualizations. Chapter two is an evaluation of other block-programming analytics research that has been done in recent years. Chapter three discusses an overview of StarLogo Nova's new data analytics tool. Chapter four details implementation steps and challenges encountered during the development of the tool. Chapter five describes the results from the visualizations generated from our analytics tool. Chapter six then lists the takeaways from our analysis, discusses the tool's impact, describes its limitations, and suggests a number of user interface modifications backed by the tool's analysis. Possible future extensions to this project are also discussed.

Chapter 2

Evaluation of Current Block-Programming Analysis Tools

In this chapter, we describe the data analysis and statistics work accomplished within the past few years in the block programming space. Specifically, Rita Chen's M.Eng thesis on ScratchStats, Benjamin Xie's SuperUROP paper on App Inventor, and Xie's M.Eng thesis on App Inventor users are discussed [24] [15] [25]. We then detail how their work can and cannot be applied to our StarLogo Nova analytics tool task given our current constraints.

2.1 ScratchStats

Scratch is an online block programming community that users can use to create stories, games, animations, and other applications. ScratchStats, an extension of the Scratch website, displays aggregated user and project data. According to Rita Chen's M.Eng thesis, projects also contain remix trees to visualize how users have built upon each others' projects [24]. The data visualized on the ScratchStats page dates back from 2007 to the present [10]. User data on ScratchStats include visualizations of the following information about Scratch:

- Previous month's website traffic

- Monthly activity trends, like new comments, projects, and users
- Monthly active users
- Age distribution of users
- Which countries Scratch users are located

These visualizations give interesting insight into the breadth of users who use Scratch. Counting monthly active users is one way that Scratch measures success and engagement on their platform.

In addition to visualizing user data, Scratch offers more project-specific data visualizations:

- Projects shared, users registered, comments posted, and studios created
- Remix trees [24]
- Monthly project shares, monthly comment activity
- Tree map of Scratch block usage from a random sample

As briefly mentioned earlier, Scratch, like StarLogo Nova, has a feature called 'Remix', which is how users share projects, make copies of other projects, and build on top of them [24]. The purpose of this is to encourage collaboration between and creativity among users of the platform. These visualizations give insight into how users use the Scratch platform. Additionally, the tree map pictured in Figure 2-1 displays how often the different Scratch blocks are used across a random sample of projects. Hovering over the rectangles on the chart gives users more detail about the appropriate block type, and clicking through yields a zoomed-in perspective of the tree map.

A major piece of ScratchStats that inspired the way we designed StarLogo's visualizations is the interactivity of the visualizations. In ScratchStats, hovering over any point on any area of the charts reveals a tooltip with more information about that particular point in the chart [10]. Some charts in ScratchStats also have togglable legends and clickable parts of the graph to give users the ability to 'tinker' with the

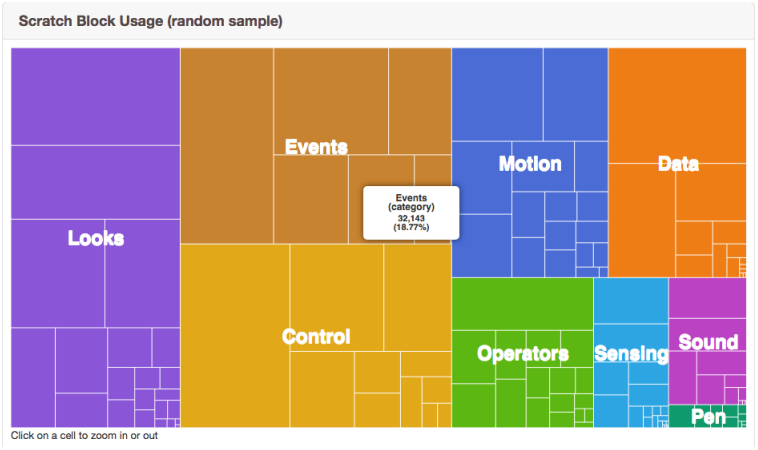


Figure 2-1: ScratchStats TreeMap of Block Usage

data [10][24]. Our analytics tool for StarLogo aims to incorporate similar interactivity in our visualizations. The hope was to ensure that the visualizations were informative and easy to understand.

2.2 App Inventor

App Inventor is a block-programming platform that is specifically used to develop mobile Android applications. On the App Inventor homepage, basic statistics on active monthly users, the number of registered users, weekly active users, number of countries reached, and the number of applications built are available [2]. From Xie's M.Eng thesis and SuperUROP paper, we see that the App Inventor team has also done some research into user progress data and grouping projects by type [25][15].

One of Xie's goals was to investigate how users grew in their computational thinking skills as they continued to use App Inventor over time. To do this, Xie used K-Means clustering algorithms to find similarities between representative users [25]. Using feature extraction and grouping, he also classified projects into different categories based on App Inventor's palette, which organizes blocks by functionality. He was thus able to gain insight on how users' projects correlate with App Inventor tutorials [15].

Although Xie's work takes a different direction from ours, we can nonetheless draw

inspiration from his thesis. His work on analyzing the relationship between tutorials and users' projects inspired us to investigate tutorials' impact on StarLogo users [15]. Additionally, Xie's analysis makes sure to only focus on active blocks [25]. This means that any blocks that did not contribute to project functionality were considered noise and removed. StarLogo blocks are similar to App Inventor blocks in that blocks can be placed on the canvas without actually contributing to the functionality of the resulting programs. Because of this, we decided to similarly filter out this noise when generating visualizations for our StarLogo tool. Further details on this process is described in Chapter 4, Implementation.

2.3 StarLogo Approach

We chose to make the visualizations interactive, similar to how ScratchStats charts are, in order to give users the option to view the data with a variety of perspectives [24]. For example, we decide to make breeds togglable for some charts so that users of our tool can compare the data between different breeds. Also, similar to Xie's work, we decide to filter out inactive blocks, which do not contribute to StarLogo program functionality [25]. This includes blocks that are disconnected or stacks that are not connected to a valid top level block. By doing this, we are able to effectively remove noise in our data source.

There is no existing data analytics tool or infrastructure to support data analysis of how users use the StarLogo Nova platform. On the site, though, any StarLogo user can open a public project and view its block code in the browser. This block data is used as our initial data source for our analytics tool. Structured in a large JSON format, this data includes the block names, positions, and how the blocks are linked to each other for every breed of a given project. The tool aims to programmatically extract this data from the server and to make it more understandable.

Unlike both ScratchStats and Xie's work with App Inventor, though, StarLogo's tool currently only works with decontextualized block data. Thus, questions of monthly activity, remix trees, and user growth over time will not be addressed in

our visualizations and analysis. We decided to choose visualizations that would convey an informative overlook of data that would give insight into the pervasiveness of tutorials, the effectiveness of breed tabs, the extent of scrolling, and block size and placement. A list of the visualizations generated by our tool is found in Appendix A. Potential ways to build upon Chen's and Xie's theses in StarLogo are discussed in greater detail in Section 6.3, Future Work.

Our goal was to create a tool that would collect and visualize StarLogo data to shed light on how to best lower the floors and raise the ceilings for users. Specifically, we wanted to address questions about the relationship between projects in the database and tutorials, about how breed tabs affect usability, about how the need for scrolling impacts usability, and about how StarLogo blocks' sizes and positions influences user experience. With the inspiration garnered from ScratchStats and App Inventor, we have designed a tool that aims to achieve this goal.

Chapter 3

Analytics Tool Overview

The following chapter describes our goals as well as the high-level constraints and challenges we encountered. A discussion on the StarLogo Nova Data Analytics tool interface follows.

3.1 Goals

We want to gather information on how to improve the StarLogo Nova platform, particularly in raising the ceilings and lowering the floors for users of many levels of expertise. In order to accomplish this, we need to understand how the platform is used. We aim to investigate the factors mentioned in Chapter 1, Introduction (Tutorials, Scrolling, Breed Tabs, and Block Size and Placement) and to understand these factors' impacts on usability. Thus, we decided to build a data analytics tool that collects, transforms, and visualizes StarLogo block data so that we can better achieve these goals.

3.2 Constraints and Challenges

There are two substantial factors that affected how we went about achieving all of the aforementioned goals. The following sections discuss what constraints and challenges we encountered that led us to follow more creative methods of collecting data and

drawing meaningful information from said data.

3.2.1 Public Projects Only

Projects on the StarLogo Nova platform can be either private or public, a setting configured by end-users themselves. For this thesis, we only have access to public data, and thus, our conclusions are drawn from public project data alone. This public project data is accessible through an HTTP GET request, which the platform uses every time it renders a StarLogo project. The data gathered and reported in this thesis is equivalent to the data one would find from opening the public projects available on the website and looking at their blocks.

Only having access to public projects introduces some constraints. Public project block data does not possess any information about the users who created the projects, so we will be making analyses from a decontextualized perspective rather than from the context of users. There is also a chance that public projects are different from private projects in some significant way. For example, public projects may look more complete, organized, or interesting if users decide to only publish their 'better' projects publicly [1]. If this is the case, our conclusions will be constrained to analyzing 'good' projects only.

Our analysis may not be an accurate representation of all projects due to selection bias. Despite all of this, we can still find useful data from these public projects nonetheless. We can still gain much insight into StarLogo by analyzing and comparing public projects' blocks and breeds.

3.2.2 Browser Limitations

As mentioned previously, the data source for our analytics tool is the data received as an HTTP response to a GET request, which is the same way that blocks are loaded onto the StarLogo platform in the browser. The current StarLogo code that loads block data runs in the browser, which means our tool must also run in the browser

[1] On the contrary, we see in Chapter 5 that the public projects in our data source are often incomplete, disorganized, and duplicates of tutorial projects.

in order to collect this block data. Given our dependence on HTTP requests and the browser, the tool needed to use the web browser for extracting and transforming the data. Our tool's performance can be impacted when it is given a larger amount of data. For example, it takes around 7 hours to query the server for all the projects in the database. Also, it takes about 1 minute to render the SVG visualizations for over 8,000 projects. In practice, the tool is intended to be used for full database querying relatively infrequently, so the amortized time cost of using our tool is about 1 minute.

Our initial plan was to create a simple script to query for and process the StarLogo block data. However, because of Cross-Origin Resource Sharing (CORS) rules, we needed to build a separate server that acted as a proxy between our querying code and the StarLogo server. We had also wanted this tool to specifically be for StarLogo researchers, not the general user body. Therefore, this analytics tool is an external web-based tool. It is not an extension of any part of the StarLogo Nova website at this time. Working with these browser limitations makes this task less straightforward, and hence, more technically intriguing.

3.3 Interfacing with the Data Analytics Tool

StarLogo's new data analytics tool is a web-based tool that extracts data from the StarLogo server and then visualizes it. Using our tool involves a few manual steps that will be outlined in the following subsections.

3.3.1 Extracting Data

The first step is to adjust the configurations such that the tool knows how many samples to query from the database. As mentioned previously, it takes around 7 hours for our tool to query for around 400,000 project samples, which is approximately the size of the current database that is loaded into the Beta server. More details about what the Beta server is follows in Chapter 4, Implementation.

After the tool finishes running, the data is manually extracted by removing the string out from either the browser's LocalStorage or IndexedDB, depending on where

the data was stored, which depends on the size of the stored data. This data is then saved locally and the researcher using the tool can then run the *parseData.py* script. This script creates *.txt* and *.csv* files needed for the visualization step and takes less than a second to run on about 69,240 project samples. This is the number of public projects found after querying for 400,000 projects.

3.3.2 Visualize Data

After the *parseData.py* script is finished running, change the configurations for the tool such that the tool knows that it is tasked to visualize the data. Charts are then rendered on the browser, visualizing the data source specified by the configurations. At this time, the configurations allow the tool to visualize a representative project for each of the top 200 most popular project signatures. The tool can also visualize a representative project for each of 10,000 random signatures not in the top 200. Project signatures are discussed in greater detail in Section 4.2: Transforming Data: Finding Duplicates. In Section 4.4.1 Top 200 and Random 10,000, we explain why we chose to visualize the top 200 most popular project signatures and a random sample of 10,000 project signatures not in the top 200.

3.4 Recap

Without much data analysis infrastructure available with StarLogo now, it is unclear to researchers how users' projects relate to tutorials, how effective breed tabs are, how users feel about the need for scrolling on the development page, and how visually overwhelming the blocks are to users. These are four key aspects we suspect to have substantial impact on both beginner and advanced users. With our dependency on the browser, collecting and visualizing project data becomes a more technically interesting challenge.

Creating data analysis tools have had lower priority in previous years when compared to other feature requests and bug fixes. However, given that StarLogo Nova had just underwent such a huge overhaul from Flash to JavaScript, this thesis takes on the

opportunity to provide greater visibility of the StarLogo platform to researchers than they have had before. Building a tool that collects and visualizes project block data can provide an incredible amount of information that would bring insight into how users use StarLogo as a whole, setting the foundation for more data-driven platform design decisions in the future.

Chapter 4

Implementation

This chapter discusses implementation details and technical challenges encountered during development. Limitations of our implementation are also discussed. Figure 4-1 illustrates the general architecture of the system, both when it is extracting data and when it is visualizing the data. Our tool accesses data from the Beta server, which holds a clone of the live database from October 2016. The Beta version of the site is the new version of StarLogo that is currently being developed.

4.1 Extracting Data

To request data from the Beta server, our analytics tool's client sends a request to its server for a predetermined amount of data. Then, the tool's server sends GET requests to the Beta server. The Beta server responds with string responses that hold

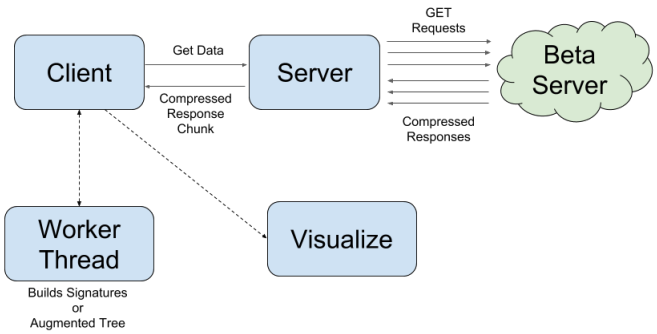


Figure 4-1: Overall System Architecture

the project data. Our tool's server then sends this to our client, which reads and transforms the data. This process repeats until the client has received the desired number of data samples (the desired number of projects). In Figure 4-1, the multiple arrows between our server and Beta server represent how the server sends chunks of GET requests to the Beta Server at a time.

Avoiding DDOS

StarLogo uses Amazon Web Services (AWS) cloud computing services [1]. Because our method of getting data is similar to a DDOS attack on the StarLogo Nova Beta server, AWS automatically shuts down my connection if I try to send a couple hundred of GET requests immediately after another. To fix this, I chunk the number of GET requests by chunks of 100, restarting the connection after every 100 GET requests to the Beta server. The size of the chunks are parameters that can be changed as a configuration setting.

Asynchronous Manipulation: Using Promises

Because sending HTTP requests are asynchronous, chunking the GET requests was a bit trickier than writing a simple for-loop. Because of ECMAScript 6 Promises, the functionality was achieved through a series of Promise functions. These functions involve pre-processing the responses received from the Beta server too, making this data collection step a little more complex.

4.2 Transforming Data: Finding Duplicates

Duplicates of IPWIT, Project GUTS, and other tutorial projects could appear in our visualizations many times because many users are introduced to the platform through a standardized program. Tutorials are generally how users are introduced to this platform, so we would expect a fairly significant percentage of the projects in our samples to be similar, or even exact duplicates. This is especially since tutorials provide very specific directions, so users are oftentimes following these step-by-step.

These duplicated projects would only reveal how tutorial-developers (i.e. StarLogo researchers) use StarLogo, not how the general body of StarLogo users use it. StarLogo Nova's 'remix' feature allows users to copy another project and build on top of it, which is another possible reason that many of our data samples may be exact duplicates of other projects. Duplicates can be seen as a source of noise in our data. With this in mind, we implemented a way to detect when different projects are so similar that they should be considered duplicates.

4.2.1 Active Blocks and Functional Duplicates

Our analysis is done on projects' active blocks only, similar to Xie's work in App Inventor [25]. Active blocks are top level blocks that are connected to other blocks, and other blocks that are connected to a top level block. Inactive, disconnected blocks are not compiled into the program and thus do not contribute to the program's functionality. Figure 4-2 pictures the types of blocks that are considered top level blocks, but none of those blocks alone are active because they are not connected to other blocks. On the other hand, Figure 4-3 shows examples of active blocks.

We define 'Functional Duplicates' to be projects whose active blocks are in the same order, have the same structure, and have the same frequency. Because of this, our analysis will consider two projects different even if they look fairly similar. In Figure 4-3, we see that the stacks have similar block types, but they do not have the same order. Our tool recognizes each of the stacks in Figure 4-3 as different. One may argue that these stacks are similar enough to be considered duplicates, but measuring similarity between our data samples requires something like a tree edit-distance algorithm. However, because implementing such an algorithm is beyond the scope of this M.Eng project, we chose to preserve the differences between stacks even if they do not seem very significant.

Our tool's data processing functionality can be built upon for any future work in measuring similarity. If our tool had combined projects that 'looked similar', we would have lost some information about the original data. Our more conservative approach of preserving these seemingly minute differences builds the foundation for

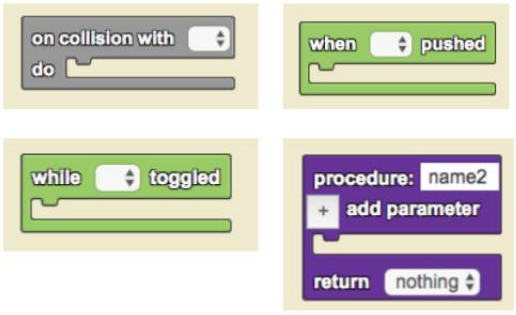


Figure 4-2: Top Level Blocks: No 'Before' and no 'After' connector

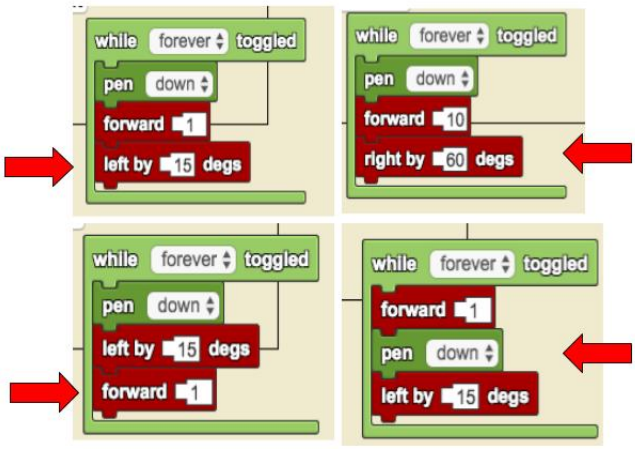


Figure 4-3: Examples of Different Stacks

future investigation into measuring similarity between projects.

For the rest of this thesis, references to blocks in regard to our data analysis refer to active, connected blocks as described in this section.

Building a Tree Structure

A data sample received from the Beta server is a very large, complex tree of both active and inactive blocks representing one project. Ignoring inactive blocks, we create tree structures for each project, where each breed and its blocks is a subtree of the larger project tree structure.

To filter out the inactive blocks, the tool iterates through the decompressed, de-

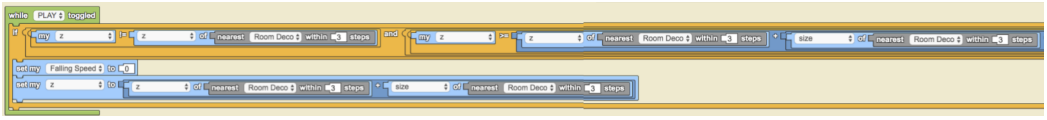


Figure 4-4: Partial Screenshot of Another Wide Stack

coded project data to build a tree structure that consists of only active blocks. More concretely, project A would be a functional duplicate of project B if they both have the same number of breeds and between their corresponding breeds, they use the same stacks of blocks with the same structure. Functionally duplicate projects do not need to have breeds with the same names.

We define a 'child' block to be a block that is connected to the current block's nested socket, or a block that is connected to its 'after' socket. For example, the 'forward' blocks in Figure 4-3 have a 'before' connector and an 'after' connector. In the top left stack of Figure 4-3, the 'forward' block's child is the 'left by _ degs' block while in the bottom left stack, the 'forward' block does not have a child. Blocks can also have arguments, which we distinguish from children because arguments tend to contribute to width calculations more than children do. Figure 4-4's wide stack shows an example of a block with many nested arguments. Argument and children blocks are represented visually differently on the StarLogo platform, and so too in the tree structures we create.

Our tool peruses through the block data, filters out the unused blocks, and builds up a tree consisting of active root, children, and argument blocks. The final tree structure looks like the following:

$$BlockName : [[child_1, child_2, ...],$$
$$[arg_1, arg_2, ...]]$$

where the first element of the array is an array of the block's immediate children, and the second element in the array is an array of immediate arguments. The array data type was chosen in this case because we want to ensure the ordering of block names are standardized across the different trees built for all the different projects.

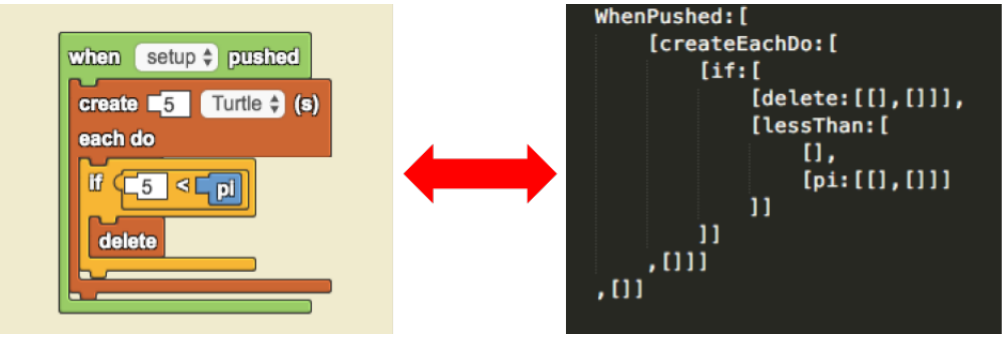


Figure 4-5: Tree Structure Example

Figure 4-5 shows a side-by-side comparison of an example stack of blocks and its corresponding tree structure. Note that we also ignore literal values, like the '5's in Figure 4-5, in our tree structure.

Signatures

After constructing the tree structure of all the active blocks in a project, the tool keeps a stringified copy of the project's tree object as a 'signature'. When the tool is in 'Extract' mode, it builds projects' tree signatures from the data and then updates the tool's internal hashmap of project signatures. Every key in the hashmap is a tree signature, and every value is a list of project IDs that have that particular signature.

When the tool is in visualization mode, we input the IDs of different project signatures into the tool via *parseData.py*. Instead of building a signature, the tool builds an 'Augmented Tree' that contains the tree structure signature as the backbone. The additional information added to this backbone includes the project's stacks' positions, block heights and widths in pixels, breed names, and other pieces of data. The 'Augmented Tree' contains data that are useful for visualizations, but are not used to distinguish similarity between projects and thus are not included in 'Extract' mode's signature creation. Because this process gives us access to project IDs, we were able to also manually opened some projects on the StarLogo site when testing our tool's accuracy and when observing outlier projects.

4.3 Mining Data

As mentioned in the introduction of this thesis, we initially went about investigating how to raise the ceilings and lower the floors for StarLogo users by thinking about tutorials, scrolling, breed tabs, and block sizes and placement. This played a large role in our decisions on which statistics were useful to gather and which were not.

Project wide information, such as the number of breeds and the number of blocks in a project, contributed to building an overview of how projects looked regarding tutorials and breeds tabs. Finding information about the position of block stacks and their widths and heights in pixels were useful in determining how users used screen real estate. Additional information, such as the number of widgets in a project, was included in our analysis as a way to see how far projects have deviated from the default blank projects, thus aiding in measuring how 'creative' users are with the platform, and how accessible it is for new users to grow to be expert users.

Complexity of block-based programming projects are multidimensional- there are many aspects that contribute to the complexity of a given project. This could include the number of blocks in the project, frequency of certain block types, number of nested blocks in the stacks, number of breeds, and other factors. For this project, we use the total number of blocks in a given project as a general measure of overall complexity. For example, we wanted to see how stack heights and widths change based on the increasing number of project blocks.

Throughout iteration, we learned that averages were not as informative as focusing on the extreme cases. Averages, like average block widths or average project block lengths, were not as effective in characterizing projects as expected. For instance, visualizing the maximum width of a stack of blocks gives more information about the complexity of a particular stack than would visualizing an average width for all stacks. So, in our final implementation, we opted for visualizations that highlight outliers- minima and maxima- as well as charts that display trends. We could have averaged the x-coordinate positions of all stacks in a given project and plotted those averages. Instead, we plot every stack's top level x- and y-positions so that we could preserve

information about each stack's position and top level block name. This alternative approach provides richer information than averaging out all of the stacks' x-positions.

Manual Work

The block data itself does not store any information about blocks' pixel details. So, to collect data on block pixel widths and heights, we needed to manually measure the blocks and manually test how blocks dynamically resize when connected to others. Alternatively, we could have used ScriptBlocks, StarLogo Nova's block library, to load the blocks visually and then get the pixel measurements after the blocks have been loaded. However, loading and rendering each project and then measuring each block with ScriptBlocks is a very time-consuming task, much more time-consuming than using pre-measured values. Also, pixel measurements for each block do not need to be calculated every time the tool was run since they would only change if ScriptBlocks changed, which does not happen often. We therefore used the Page Ruler Google Chrome Extension to manually measure each of StarLogo's blocks' heights and widths [16]. This method is sufficient for our needs, because our goal is to gain an improved, bigger picture understanding of how users use the platform as a whole. We do not require much precision in our pixel measurements.

4.4 Visualizations

We used an open source data visualization library called D3, version 4, to output graphs for this analytics tool [5]. This is a JavaScript library we used to generate scatter plots, box and whisker plots, circle graphs, and other types of charts specifically for the StarLogo Nova data. These charts include togglable legends and hovering tooltips, in an effort to make the visualizations interactive and informative. Alternative software, such as a popular desktop data visualization application called Tableau, could also be used for visualization once data is extracted [13]. For our purposes, we decided to continue forward with D3 since it is not a desktop app, but instead a library that programmers can use to write customizable graph-generating code. Using

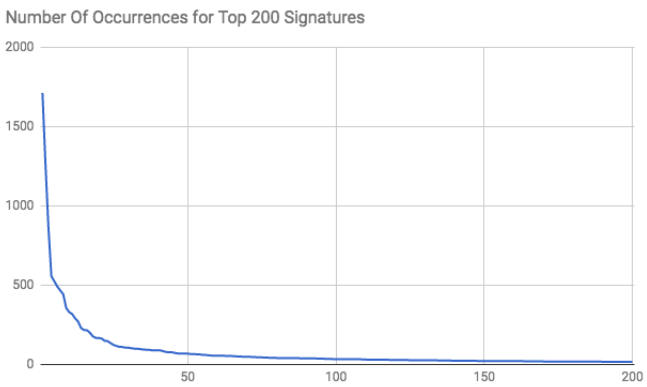


Figure 4-6: Duplication rate of top 200 project signatures

D3 ensures that future users of this analytics tool can make changes and updates easily.

4.4.1 Top 200 and Random 10,000

Ideally, we would visualize a representative project from each project signature in the entire database. However, we ran into RAM and performance issues when attempting to visualize every single data point. In this thesis, we visualize the top 200 most popular project signatures and a random sample of 10,000 signatures that are not part of the top 200 most popular signatures.

We wanted to see how the most popular projects in the database looked, so we stepped through a list of signatures sorted by decreasing number of occurrences until any new signatures stopped enhancing our understanding of the most popular behavior in the database. This was determined to be at the 200th project signature, since after that point, the signatures had very few occurrences and hence stopped representing 'popular' behavior of the database. Figure 4-6 shows what we found for the top 200, where the X-Axis represents the n^{th} most popular project signature and the Y-Axis is the number of occurrences in the database. The number of duplicates per signature is inferred to decrease in a predictable manner for signatures after the 200^{th}. We determined that the 200^{th} most popular project would be the threshold between signatures that represented 'popular' and 'not popular' behavior.

For our random sample, we wanted to have a sample size that captured the be-

havior of the signatures not part of the top 200 project signatures. We chose a sample size of 10,000 because it was the largest sample size we were able to take before reaching performance limits, which are discussed in the next section. Additionally, we saw that, after repeatedly visualizing random 10,000 samples of our project signatures, we would consistently receive qualitatively similar results.

Visualizing both the top 200 project signatures and a random sample of 10,000 project signatures not in the top 200 therefore gives us a close approximation of the behavior of all of the projects in the database. That being said, the number of popular projects to visualize and the random sample size can both be re-configured in the tool.

4.5 From Extraction to Visualization

As briefly mentioned in Section 3.3 Interfacing with the Data Analytics Tool, running our tool involves a few steps. These are discussed in more technical detail here.

1. **Set Extract Mode** In this step, the tool requests data from the Beta server and builds tree object signatures from this project data. The $MODE$ configuration in *common.js* is set to $Extract$. In this mode, the tool stores a hashmap with $\{treeSignature : [ID_1, ID_2, ...]\}$ key-value pairs. This hashmap is stored in either LocalStorage or IndexedDB, depending on the size of the data, and is updated until the tool has finished querying and processing all the data from the Beta server.

2. **Manually Load Extracted Data** The data is stored in the browser either in LocalStorage or IndexedDB as a string and must be loaded locally as a *.txt* file. This can be done with manual copy-and-pasting from the browser's LocalStorage interface, or by running a few commands on Chrome console to copy the string from IndexedDB. In our tests with public project data, LocalStorage suffices.

3. **Run Python Script** *ParseData.py* is a Python script that parses the *.txt* file that was created manually in Step 2. This script iterates over the tree

signatures, counts the frequency of each signature, and chooses a representative project ID for each signature. This information is generated into a *.csv* file. Also, a list of representative IDs for the top 200 project signatures and a list of representative IDs from 10,000 random signatures not in the top 200 are generated into *.txt* files. These files are needed for visualization. Each ID that is added to the *.txt* files represents a different signature in the database, so there are no duplicates in our visualizations.

4. **Visualization Mode** After the necessary *.txt* files are generated, the tool's *MODE* configuration in *common.js* needs to be set to either 'VIZ_200' or 'VIZ_10K', which visualizes the top 200 project signatures or the random sample of 10,000, respectively.

5. **D3 Graphs Rendered** All graphs are loaded onto one page, one after the other. Scrolling down the page reveals the different charts. This was done so that all the visualizations are available at once for the user. This format is borrowed from ScratchStats, which also shows all of its visualizations on one page [10].

Limitations

As a brief benchmark, running the tool to visualize 10,000 random project samples takes around 15-20 minutes. During the development of this thesis, this running time was the longest we could work with. D3 uses SVG for graphs, so as the data set size increases, the laggier rendering becomes. Tooltips that appear when hovering over certain data points are also slow to render at large sample sizes. Ways to improve this can be addressed in future iterations of this tool, especially if it becomes an extension to StarLogo Nova for the general body of StarLogo users.

Also, because this tool involves a multiple step process of extracting and then visualizing the data, and because the data transformation depends on the browser, the tool requires a researcher to take a few manual steps to get meaningful data. In future iterations, data extraction from the browser and automating *.csv* and *.txt* file

generation could improve the tool's usability.

Chapter 5

Results

In this chapter, we showcase examples of the data visualizations our tool generates, representing data from the top 200 most popular project signatures. We also include visualizations generated from a random sample of 10,000 project signatures not in the top 200. The most popular signature occurs 1712 times and represents an empty project with three breeds. The 200^{th} most popular project signature occurs 18 times and is an IPWIT tutorial project remix.

As mentioned in Section 4.4.1 Top 200 and Random 10,000, we visualize the top 200 most popular signatures and 10,000 other random signatures to give us a sense of what the entire database looks like, without actually visualizing the entire database and without crossing our performance limits. We include visualizations of the top 200 and the random 10,000 side-by-side as a way to compare the behavior of the most popular projects with that of the less popular projects. In doing so, we aim to achieve an accurate overview of the entire database's project behaviors.

We have reached the conclusions in this section by analyzing the tool's generated data visualizations, manually opening the representative projects from each of the top 200 popular projects, and manually opening large project outliers. Manually opening a project involves using the project ID to view the project on the StarLogo Nova platform. The pie charts included in the following chapter were also generated manually after running our analytics tool. Given our tool's capability, we ask the follow questions about our data:

- **How do the StarLogo projects look as a whole?**

  Our analysis involved filtering out inactive blocks from our project data because they were a source of noise. We wanted to know how the database looked after filtering these out. We were especially curious since the StarLogo team did not previously have an answer to this question before.

- **How similar are projects to tutorials? How different?**

  As mentioned in Chapter 1, Introduction, most users learn StarLogo with tutorials from standard programs like IPWIT or Project GUTS. We wanted to know how similar projects were to tutorials, or how different the projects were from tutorials. We wondered how users expanded their projects away from tutorials, if they did. These insights could shed light on how well StarLogo has been able to encourage creativity for both new and expert users.

- **Are breed tabs organizing users' code effectively?**

  Visually similar to web browser tabs, breed tabs can only be viewed one at a time and aim to keep different agents' blocks separated, in order to make the blocks more manageable. Although different from typical breeds, the two special breeds 'Everyone' and 'The World' are visually represented as tabs, just like any other breed. Our analytics tool can reveal how 'Everyone' and 'The World' are used by users of all levels of expertise, and reveal if the tab layout actually organizes blocks effectively.

- **What is the extent of required scrolling on the development page?**

  Earlier, we expressed interest in learning how scrolling on the development page (i.e. scrolling between Spaceland and the canvas) may impact usability. We also discussed how StarLogo block stacks' sizes and positions can potentially be overwhelming. With our visualizations, we observe how much the page's layout, block stack sizes, and block positions contribute to the development page's required amount of scrolling. Then, we infer from our data if both beginner and advanced users are possibly limited by this amount of scrolling.

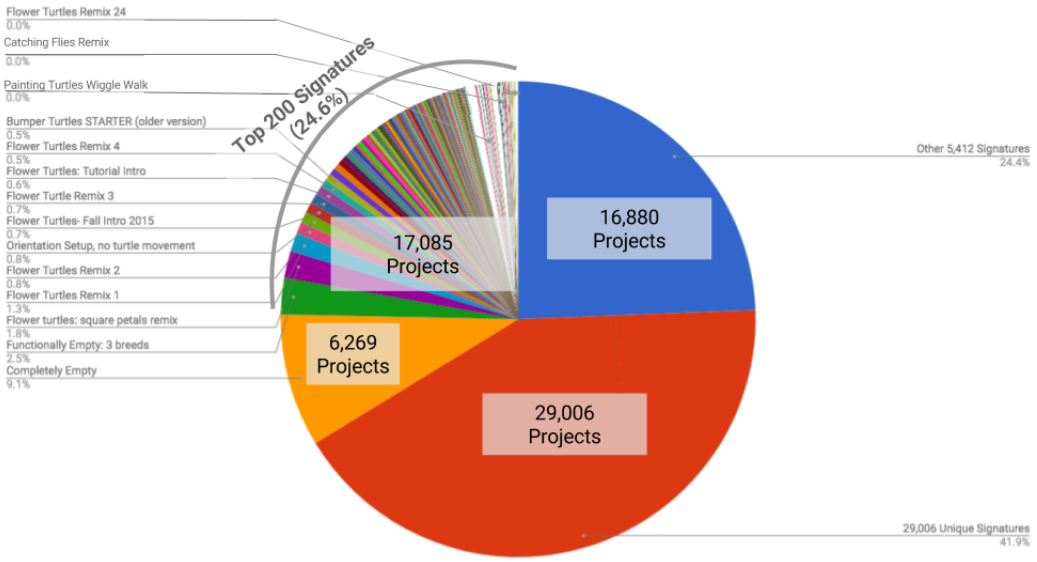


Figure 5-1: Entire Database Categorized by Signature

5.1 How do the projects look as a whole?

We queried approximately 400,000 projects from the Beta server and received data on 69,240 public projects. After processing this data, we found that these 69,240 projects yield 34,618 different tree object signatures, not including the Completely Empty project signature. This means that 50% of the project data we received are copies of some other project in the database. The 69,240 projects represent only 34,618 functionally different projects.

In Figure 5-1, we see a pie chart that represents the percentage of projects in the database that have the indicated signature types. 24.6% of the database, that is 17,085 projects out of the 69,240 projects, share the top 200 most popular project signatures. For this pie chart, we manually opened a representative project from each of the top 200 signatures on the StarLogo site, describing each of the signatures by inspecting projects' blocks and titles. Detailed results from this are found in Section 5.2.1, Top 200 Project Signatures are Tutorials or Tutorial-Inspired.

41.9% of the database, which is 29,006 projects, have unique signatures. This means that each of these projects' arrangement of blocks only occur once in the database. 24.4% of the database, that is 16,880 projects, share signatures that are

not unique, but are not duplicated enough to make the top 200 threshold. Specifically, this 24.4% represents signatures that have less than 18 occurrences but more than 1 occurrence.

5.1.1 Empty Projects

As we see in Figure 5-1, 9.1% of the database, that is 6,269 projects, are Completely Empty. Completely Empty is defined as a project without any blocks at all. The number of Completely Empty projects is gathered from a direct filter of projects without any blocks at all. Thus, the tool does not currently have the granularity of determining how many breeds are in these Completely Empty projects. We can also see in the figure that the most popular project signature is Functionally Empty with 3 breeds- which represents 2.5% of the database. Functionally Empty refers to projects that only contain inactive blocks.

After manually opening a representative project from each of the top 200 project signatures, we found that the project signature that represents a Functionally Empty project with 4 breeds appears in the top 200 and represents 0.04% of the entire database. Taking these three types of empty projects together, we see that at least 12% of the projects in the entire database is empty in some way. There may even be more empty projects (i.e. functionally empty with a different number of breeds) among the unique signatures and the other 5,412 signatures. A significant portion of the projects in the database are empty, which may indicate something about how accessible the platform is to new users.

Note that the visualizations generated by our new analytics tool do not include Functionally Empty or Completely Empty projects.

5.1.2 Overview: Duplicated, Potentially Similar, or Empty

We have discussed a high-level view of the StarLogo projects database. An overview of what was discussed and how it addresses our question 'How do the projects look as a whole?' follows.

- Empty At least 12% of the database is empty or functionally empty, including Completely Empty projects, projects with three breeds but no active blocks, and projects with four breeds but no active blocks. This percentage does not include the empty projects that may not have been explicitly captured. For example, there could be a project in the database with a signature that represents a project with 2 breeds with no functional blocks. So, at least 12% of the database is empty.

- Duplicated Half of the database of public projects are duplicates of some other project. We found 69,240 public projects, but they represent only 34,618 functionally different projects.

- Many Unique, but Potentially Similar Over 40% of the database's projects have unique signatures- these projects do not look like any other project in the database. Also, the top 200 project signatures only represent 24.6% of the entire database. As stated previously, projects that are considered 'different' in our analysis might still look similar. This shows that there is diversity in the database, but 'how diverse?' is still an open question.

5.2 How similar/different are they to tutorials?

We are aware that most users are introduced to StarLogo through some tutorial or standardized program like GUTS. As stated in Chapter 1, Introduction, over 90% of the projects on the platform are expected to be from Project GUTS [8]. How similar are projects to tutorials? If users do explore the platform beyond tutorials, how different are the projects? We use our analytics tool to visualize projects' similarities to and differences from tutorials. In the following sections, we discuss how the projects represented by the top 200 most popular signatures look and then detail how project samples look from breed and block-level perspectives.

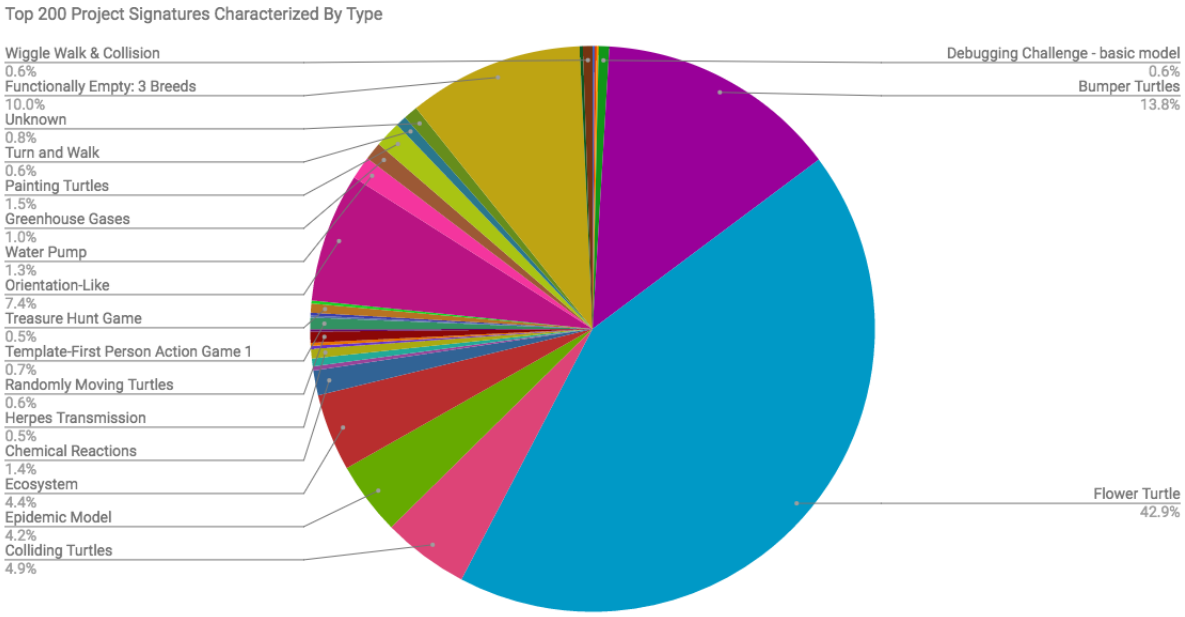


Figure 5-2: Top 200 Project Signatures Classified By Type

5.2.1 Top 200 Project Signatures are Tutorials or Tutorial-Inspired

Using the StarLogo Nova platform, we manually opened a representative project from each of the top 200 duplicated signatures and determined that they were mostly remixes of either IPWIT tutorials or Project GUTS projects. Figure 5-2 is another pie chart that shows the classification of each of the signatures in the top 200. We qualitatively characterized the signatures in the top 200 by seeing if projects shared similar titles, similar blocks, and similar behavior when the code was run.

Figure 5-2 shows that over 40% of the projects that share one of the Top 200 project signatures share a signature that is some remix of Flower Turtle, an IPWIT and GUTS tutorial project. The Flower Turtle project is a project that involves creating an agent that leaves a trail as it moves around the terrain in a flower path. Remixes of Bumper Turtles, a tutorial project from Project GUTS, is the second most shared type of project signature. A brief glance over the rest of the top 200 project signatures' representative projects indicate that most of these signatures are remixes of some other tutorial-related projects.

The 'Orientation-Like' project signature type, mentioned in Figure 5-2, look like Flower Turtle projects, which makes sense because this tutorial is called Orientation in IPWIT. We preserved this difference in the pie chart since there were a substantial number of projects who chose to title their projects 'Orientation' and not Flower Turtles, which may speak to how users were introduced to the tutorial. In other words, this could show the reach of Project GUTS versus Imagination Toolbox.

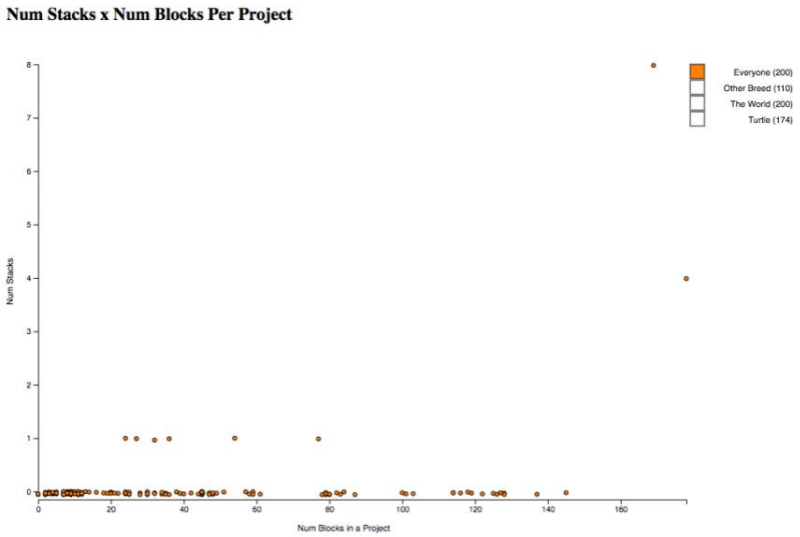
The 'Unknown' project signature type represents difficult-to-describe signatures, such as those that are almost empty, have a few stacks, or whose code has no describable behavior when run. Another interesting project title we found when manually opening representative projects was 'Herpes Transmission'. The project itself looks like a modification of the Epidemic Model project, which is another tutorial. Instead of grouping this particular project signature with the Epidemic Model category, we preserved the 'Herpes Transmission' label in this pie chart to present an example of how users are applying StarLogo. The other project categories in Figure 5-2 that have not been explicitly described in this section are known to be GUTS or IPWIT tutorials as well.

From this analysis, we realize that most of the top 200 project signatures represent tutorial-related projects, and therefore, the visualizations of the top 200 project signatures in this chapter are viewed with the understanding that they represent tutorial remixes, or in other words, early experiences with StarLogo.

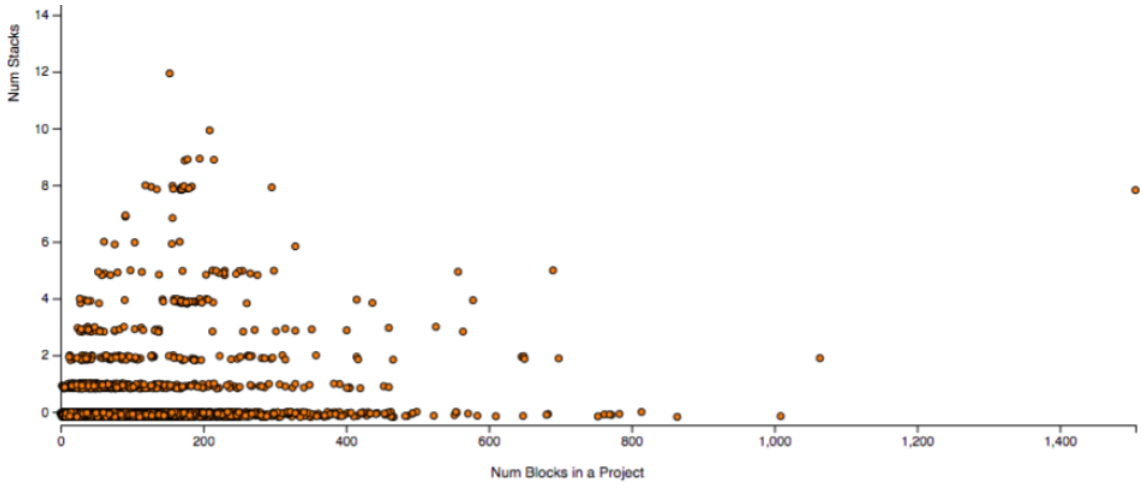
5.2.2 Breed Behavior Trends Similar to Tutorials

Figure 5-3 shows an example of one of the charts generated by our visualization tool. The X-Axis is the number of blocks in a project, the Y-Axis is the number of stacks, and the legend is togglable between different breeds. Each point represents a project under the color-specified breed. In this screenshot, we are looking specifically at the 'Everyone' breed.

Figure 5-3a is the visualization for the top 200 project signatures. Most of the points, no matter what the total number of blocks are in the project, are at 0 stacks for the 'Everyone' breed. Only 8 of the top 200 project signatures have 1 or more



(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

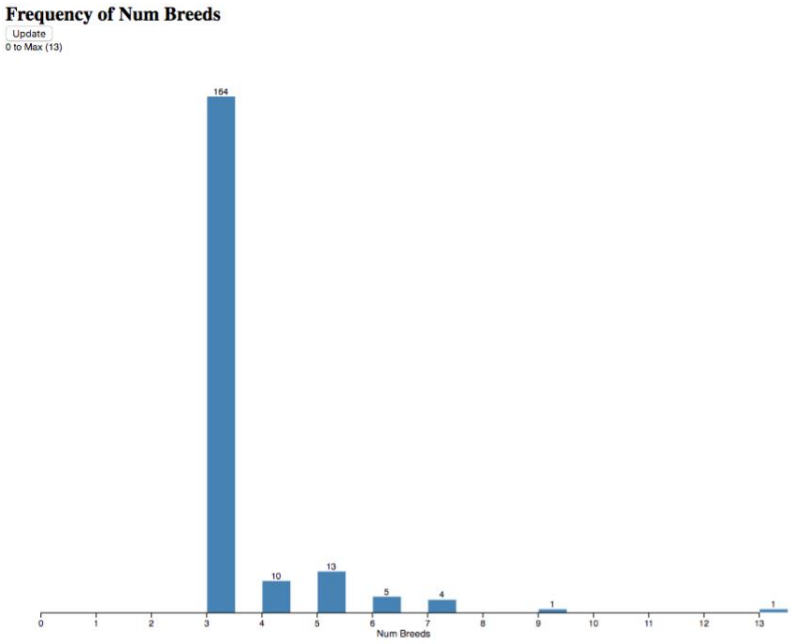
Figure 5-3: Number of Stacks vs Number of Blocks in Project in 'Everyone' Breed

stacks in the 'Everyone' breed. Having at least 1 stack means that, if the user used the 'Everyone' breed correctly, there is some shared behavior between all the agents in that project. Because the top 200 represents tutorial remixes, we can conclude that the tutorials themselves very rarely use the 'Everyone' breed, and thus, very rarely have shared behavior between their agents.

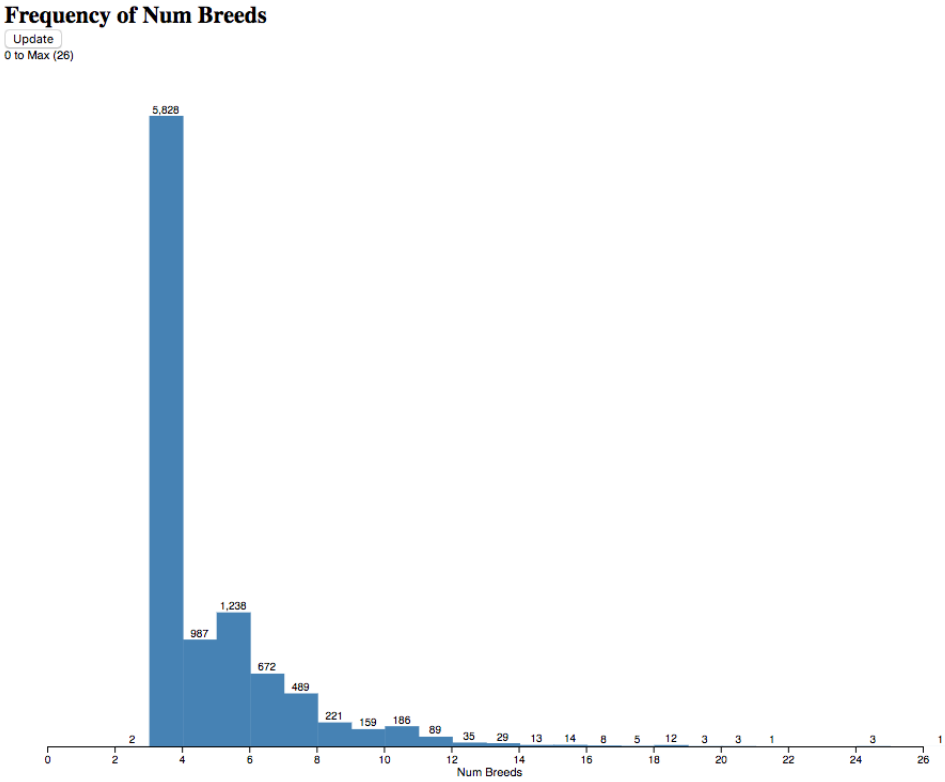
We can compare this with our random sample visualization in Figure 5-3b. We can see that a large number of projects still have 0 stacks in their 'Everyone' breed, but there are also a significant number of projects that have 1 or 2 stacks. So, a significant number of projects have 1 or 2 Procedures shared between agents, if users were using 'Everyone' the way we expected them to. Overall, this shows that projects seem to follow the behavior introduced to them through IPWIT tutorials or those from Project GUTS. We can look at another chart generated by our tool that displays users' breed behavior at a higher level. In Figure 5-4, we see a histogram that counts how many breeds each project has. In Figure 5-4a, we see that most of the projects have three breeds. This is expected, since most tutorials introduce students to three breeds, which is also the default number of breeds when a user creates a new StarLogo project. When we look at the histogram from our random sample in Figure 5-4b, we see that 3 breeds is still a popular choice. 41.68% of the projects in our random sample have more than 3 breeds, and only 2 of the project signatures in this random sample have less than 3 breeds (2 breeds). More than half of the projects in our sample keep the default number of breeds- which is similar to the behavior displayed in the tutorial remixes of the top 200 project signatures.

5.2.3 Project Sizes Slightly Different From Tutorials

In Figure 5-5, we find another chart generated by our tool: a histogram that describes the counts of how many total blocks each project in the data sample has. Figure 5-5a is an overview of the distribution for the top 200 most popular project signatures, while Figure 5-5b shows the zoomed in results from our random sample. Our tool makes this histogram interactive, as you can zoom in to see the distribution in a specific quartile.

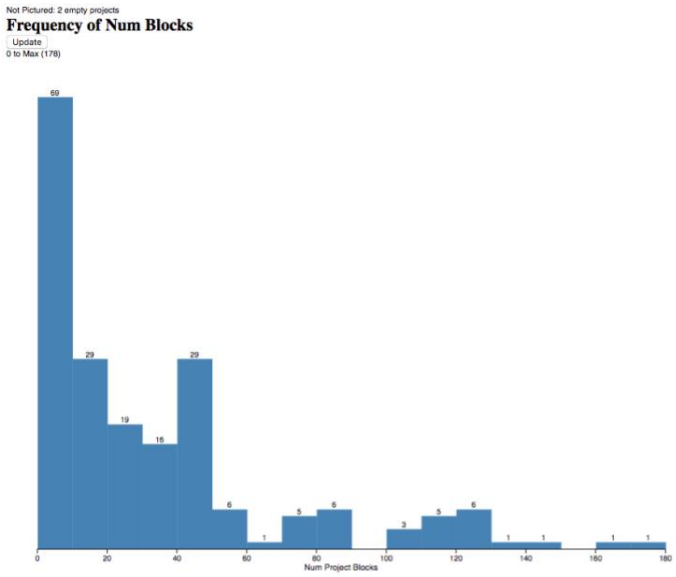


(a) Top 200 Project Signatures

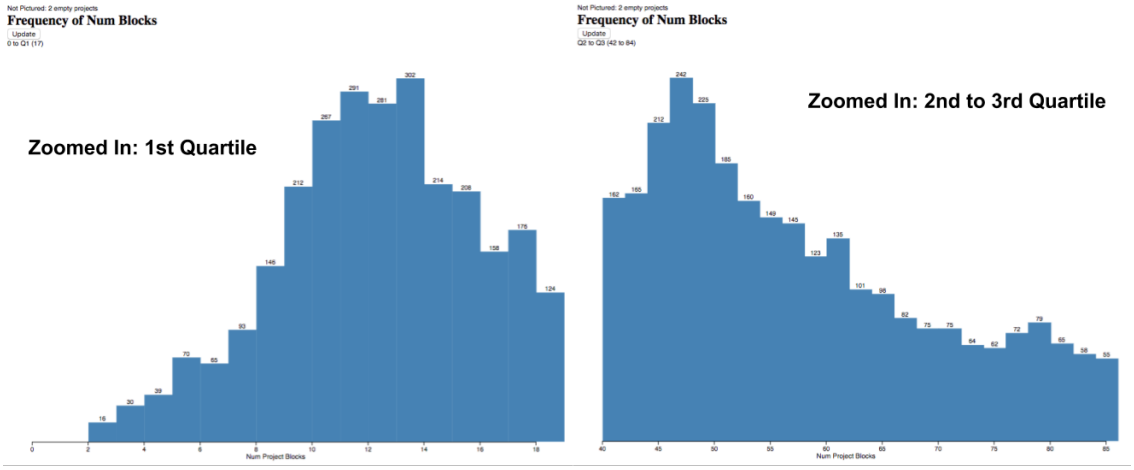


(b) Random 10k Project Signatures

Figure 5-4: Frequency of Number of Breeds



(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

Figure 5-5: Frequency of Number of Blocks

When zooming into the first and second quartiles in Figure 5-5a, we see that projects with a total of 9 blocks is the most popular, which is expected because a standard Flower Turtles project is 9 blocks large. Another popular project size is 45 blocks. Again, these numbers represent the results from the top 200 project signatures, which consist of tutorial remixes. If we compare these results to the histogram in Figure 5-5b, we see that there are a fair amount of projects with 9 blocks. However, the most popular project size in our random sample is 13 blocks. Projects between 45 and 50 blocks large are also fairly popular. In general though, we see that there is a variety of different project sizes in our random sample.

Does this indicate that users are exploring the platform more? Or does it mean users are remixing projects and only changing them a little bit? It is not immediately clear why projects are a little bigger in our random sample. Investigating this could be an interesting direction to look into for future research. Overall, we know that users as a whole are deviating from tutorials in some way.

5.2.4 Overview: Fairly Similar, Sometimes Larger

Projects in the database do seem fairly similar to tutorials, though there is a significant portion of projects that are different. The following insights about similarity with tutorials were gathered from analyzing the project data.

- Top 200 $\sim$ Tutorial Remixes The top 200 project signatures represent remixes of tutorial or tutorial-related projects. The signatures in the top 200 that are not clearly tutorial projects are either functionally empty or close to empty, thereby probably representing projects created by new users and potentially examples of users attempting to follow tutorials. A majority of projects in our random sample have 3 breeds, the default number of breeds for a new StarLogo project. This is also the most common number of breeds in the top 200 most popular signatures, which are mostly tutorial remixes.

- Breed Complexity is Similar The breed behavior between the top 200 signatures and our random sample is similar, which implies that most projects in the

database are similar to tutorials in terms of breed complexity. For instance, the 'Everyone' breed is the least complex and 'The World' breed is the most complex, both for tutorials and for the signatures in our random sample.

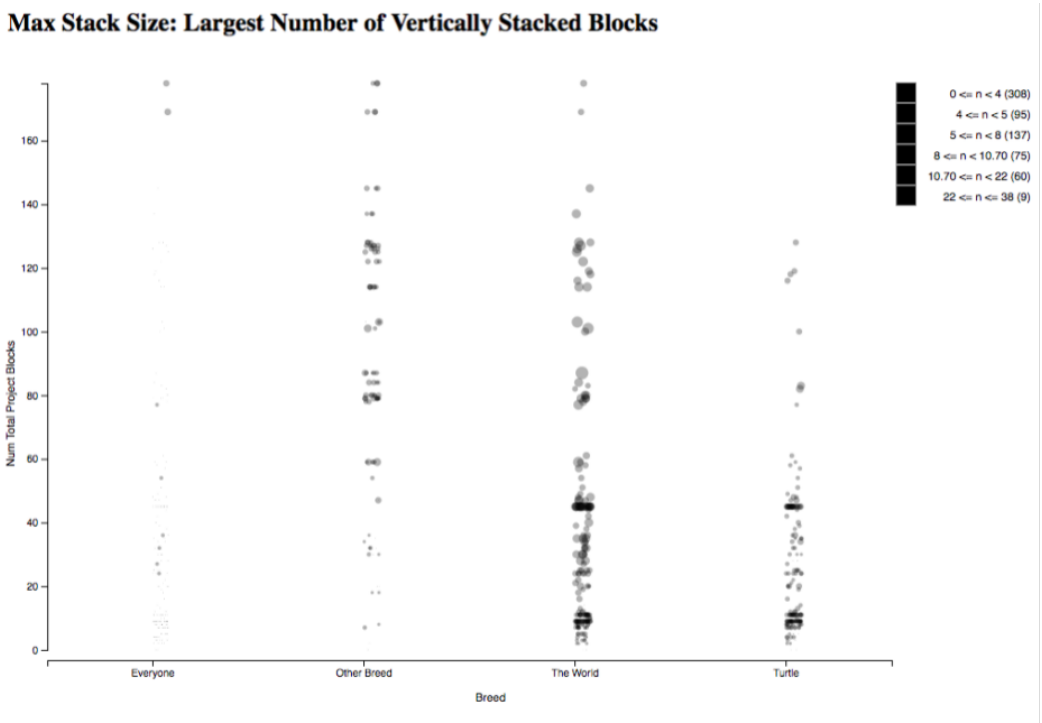
- Overall Contain More Blocks Projects in our random sample of 10,000 overall have slightly more total project blocks than those in the top 200. In general, there is large diversity in the number of total project blocks in our random sample. It is unclear why this is, but it does indicate that users are straying away from tutorials.

5.3 Are breed tabs organizing users' code effectively?

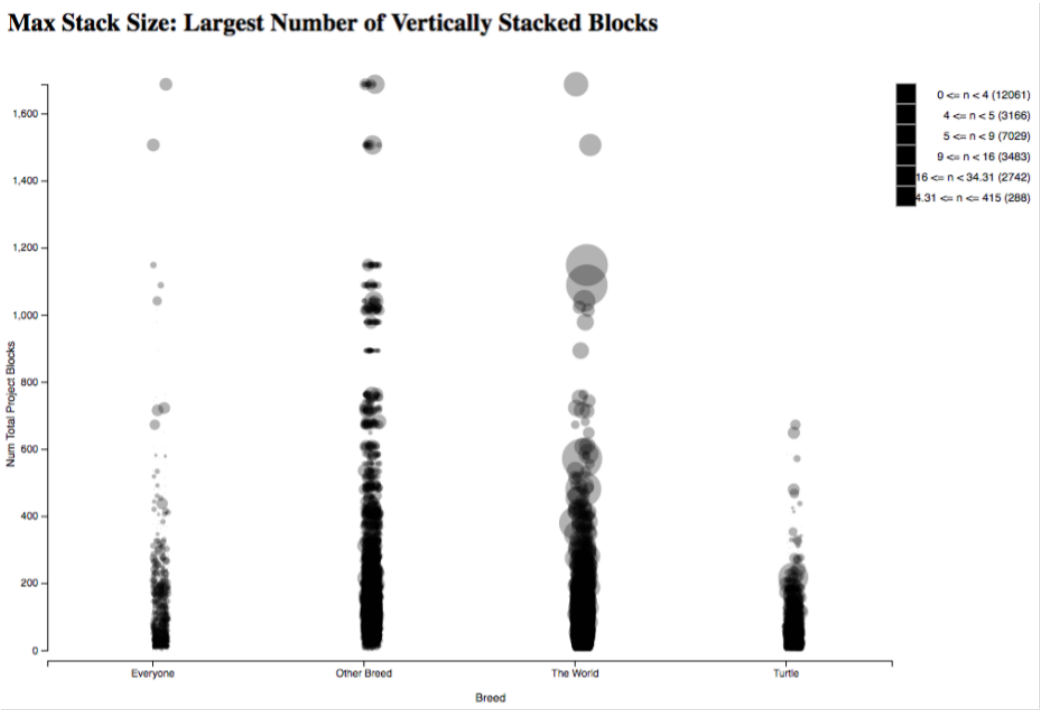
We would like to know if the way that breeds are visually displayed as tabs effectively organizes users' blocks. As a reminder, StarLogo projects have a default of three breed tabs: 'The World', 'Everyone', and 'Turtle' (a basic breed). Each breed has its own set of blocks. If 'Everyone' was used correctly, then the 'Everyone' breed across all the projects in our samples should mainly contain Procedure blocks, if they contain anything at all. This is because it is not very common for breeds to all share the same attributes, and if they do, they might not share that behavior all the time. Thus, we would expect the 'Everyone' breed to be empty or to have the code that defines these shared behaviors as Procedures, to be called by the code in other breed tabs whenever applicable. We expect 'The World' breed to have blocks in which other agents are created. We would like to know if these tabs actually help users organize their block code. The charts in the following sections give insight into how breeds are used by users, including the top 200 popular project signatures that stem from IPWIT or Project GUTS tutorials.

5.3.1 Everyone breed is rarely used

The Max Stack Size chart in Figure 5-6 is an interactive circle chart generated by our analytics tool. It shows the disparity between stack heights across different breeds,



(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

Figure 5-6: Max Stack Size Across Different Breeds

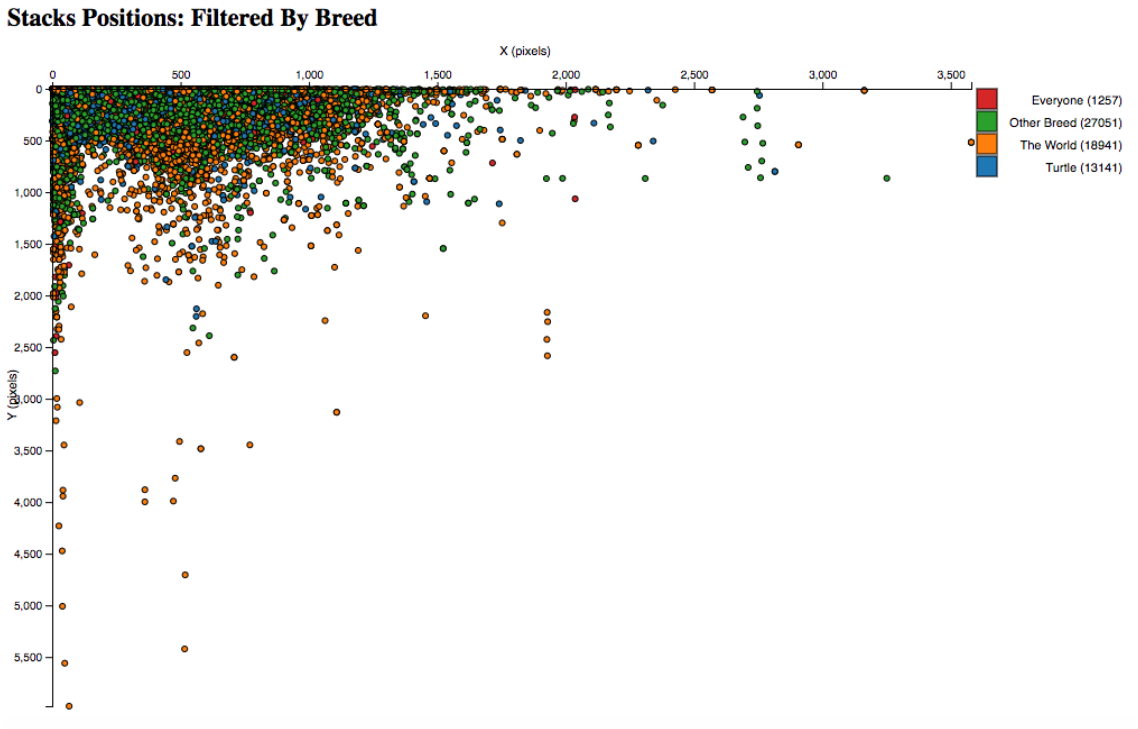


Figure 5-7: Stacks Filtered By Breed

where stack height refers to the number of blocks vertically placed in a stack. The Y-Axis represents the total number of blocks in a project. Each circle in the circle charts represents the maximum stack size for a specific project for a specific breed. The size of each circle is proportional to how many blocks are in the tallest stack of a given breed for a given project. It is clear even from just a glance that the 'Everyone' breed, both in the top 200 and in our random sample, is the least complex breed. Small and sparse circles in the 'Everyone' column show that the top 200 project signatures themselves do not use 'Everyone' very often. This makes sense, because tutorials are meant to be simple, and including Procedures into tutorials would be a little too advanced for a brand new user to understand. In general, as stated earlier, we do not expect users to use the 'Everyone' breed very much.

Our random sample shows a more diverse use of the 'Everyone' breed, but generally, it is aligned with what is expected. Figure 5-7 is a scatter plot of all the stacks in our random sample of 10,000, filterable by breed. The X-Axis is the x-position on the canvas in pixels, and the Y-Axis is the y-position on the canvas in pixels. Briefly

hovering over the 'Everyone' breed points in Figure 5-7 reveals tooltips that state what the top level block is for each point. For the 'Everyone' points, most top level blocks are Procedures, as expected. Users may be understanding the notion of the 'Everyone' breed well and are creating projects significantly different from tutorials. However, for projects who have 0 stacks in the 'Everyone' breed, it is possible that those projects have repetitive code across their breeds that would otherwise be represented as Procedure blocks in 'Everyone'. Investigating this is a possible direction for future iterations of our analytics tool.

As an aside, investigating 'Everyone' revealed an unexpected potential area of improvement for the platform. While opening projects manually on the StarLogo site, we realized that with the breed tab layout, we needed to click the 'Everyone' tab in order to see what was in it. We could not tell if it was an empty canvas just by looking at the tab, which would have otherwise been more efficient.

5.3.2 The World breed can get overwhelmingly complex

From a high level look at Figure 5-6, it is clear that 'The World' is most complex. The circles with the largest radius are most prevalent in 'The World' column for both the top and bottom charts. The tallest stack from our random sample's visualization, for example, is 379 blocks tall, which is over 40 times as many blocks as the most popular Flower Turtles project (9 blocks) in the database. The sheer number of blocks reveals just how complex projects can get.

To get a better sense of the complexity of 'The World', we can take a look at Figure 5-8. This is a scatter plot generated by our data analytics tool that shows the maximum stack height on the Y-Axis and the number of total blocks in the project on the X-Axis. Each point in this figure represents a project, specifically its 'The World' breed. The stack height increases dramatically between 0 and 20 total project blocks. The trend is even clearer in the results from our random sample. Stack heights explode between projects of size 0 and 200 blocks, and look to continue expanding as the number of project blocks increases. Stack height in 'The World' breed is overwhelmingly complex.

After manually opening of the representative projects from our random sample on the StarLogo site, we found that 'The World' was indeed the place where users added blocks to create agents. We would need further investigation into the types of blocks present in 'The World' breed in our sample to gain more concrete confirmation of how well users understand 'The World'. For instance, are people creating agents in other breeds or in the 'Everyone' breed when they should be placed in 'The World'? Again, this is a possible direction for future iterations of our tool.

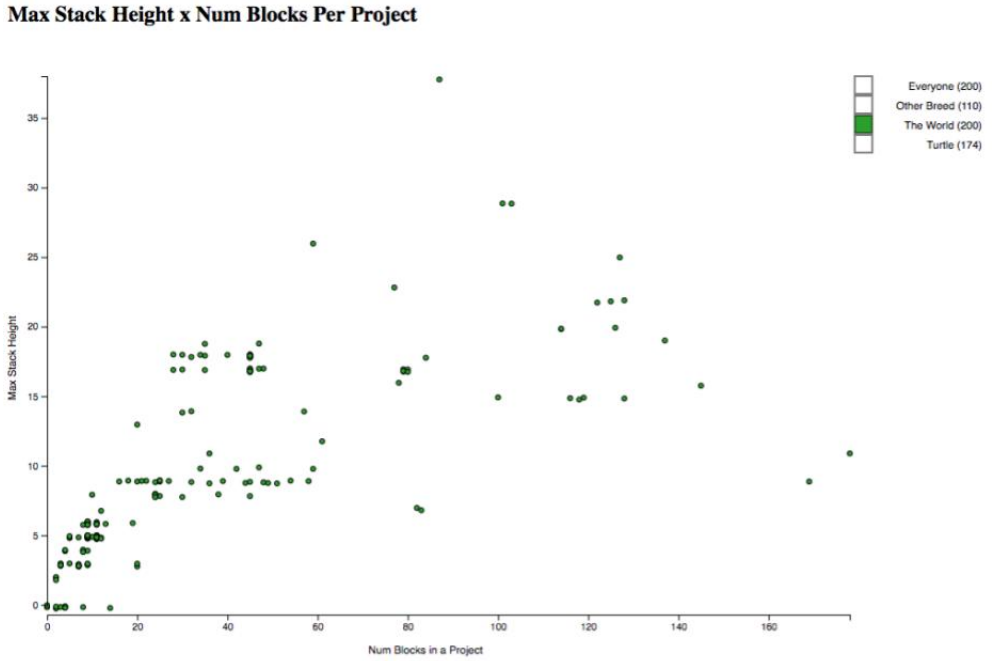
What is clear, though, is that 'The World' breed is very overwhelming and does not help make users' blocks particularly organized. Since agents and terrain elements are normally created and set up in 'The World', numerous, large stacks can be scattered all over just 'The World' itself.

5.3.3 Sometimes Too Many Breeds

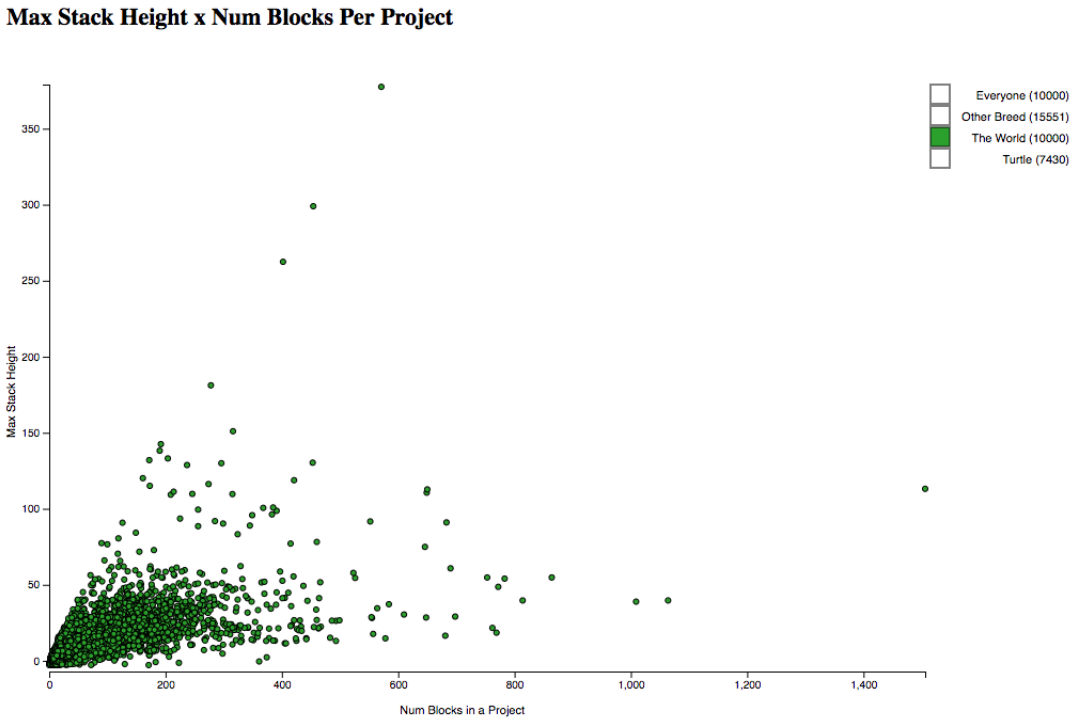
Figure 5-9 contains box-and-whisker plots generated by our analysis tool. The Y-Axis represents the number of blocks in a project and the X-Axis is the number of breeds a project has. Figure 5-9a is the visualization from the top 200 project signatures, representing the behavior of tutorial-based projects. We can see from the figure that most projects have 3 breeds, the default, with a few exceptions. This is aligned with expectations.

When we take a look at Figure 5-9b, which represents our random sample, however, we see that a significant number of projects have more than 10 breeds- some even 26 breeds. Adding too many breeds might indicate that these projects' users do not truly understand how to use StarLogo and its breeds. After manually opening some of these large projects on the StarLogo page, we found that it was visually overwhelming to see so many breed tabs on the screen. Having over 20 breeds could mean that expert users are building very complex projects, but in general, users should not be creating so many breeds. One theory is that users may be confusing StarLogo breeds with the Scratch concept of sprites, which requires users to create a new 'canvas' for every character or element of the program [9].

Regardless, the breed tabs themselves are overwhelming, since it is difficult to tell

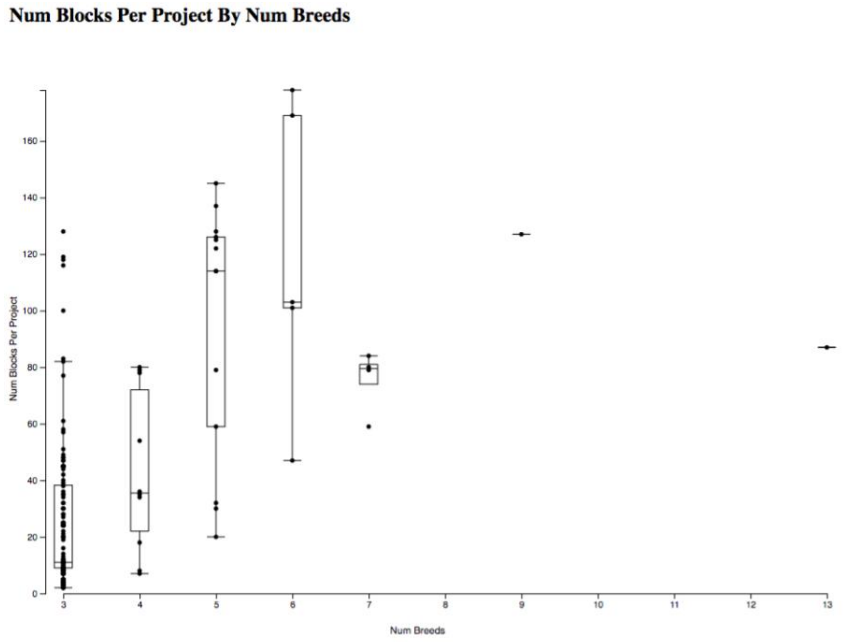


(a) Top 200 Project Signatures

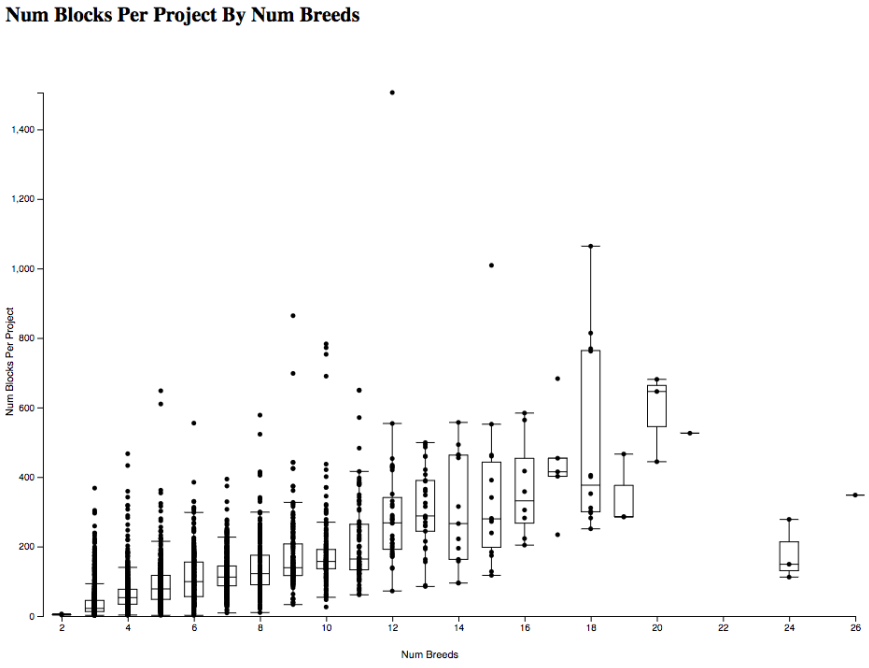


(b) Random 10k Project Signatures

Figure 5-8: Max Stacks vs Number of Blocks in Project in World



(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

Figure 5-9: Num Blocks vs Num Breeds

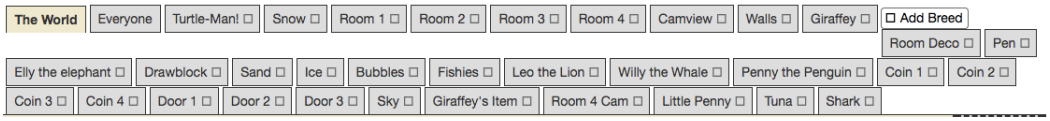


Figure 5-10: Project with Many Breeds

what each breed is and what each breed tab contains. A screenshot of a project with many breeds is pictured in Figure 5-10 which shows how difficult it is look at the breed tabs and understand what kinds of blocks are in each tab. Also, in StarLogo, some breeds are purposefully empty. Since empty breed tabs look the same as other tabs, it could be difficult to tell which of these breeds are intentionally empty and which are not.

According to Figure 5-9b, projects with more breeds generally have more blocks. So, when projects grow to this size, rendering and general site performance is greatly slowed, since there are so many blocks. If a user loses track of which breed they want to make changes to, it would be natural to click the breed tabs until they find the correct breed. However, this search method would be doubly time-consuming because rendering blocks after each breed tab switch is noticeably slower for larger projects.

5.3.4 Overview: Breeds Understood, but Visual Overload

We have gathered the following from our analysis on projects' breed behavior:

- Aligned with Expectations From a glance, users are generally using 'Everyone' and 'The World' the way we expect them to.

- Breed Tabs Overwhelming When the number of breeds increases, the breed tabs themselves become overwhelming and disorganized. This is especially overwhelming for experts, or users who are otherwise comfortable enough with the platform to add so many breeds.

- Breed Tabs are Uninformative The tab's labels do not indicate anything about what kinds of blocks and how many blocks there are in that breed.

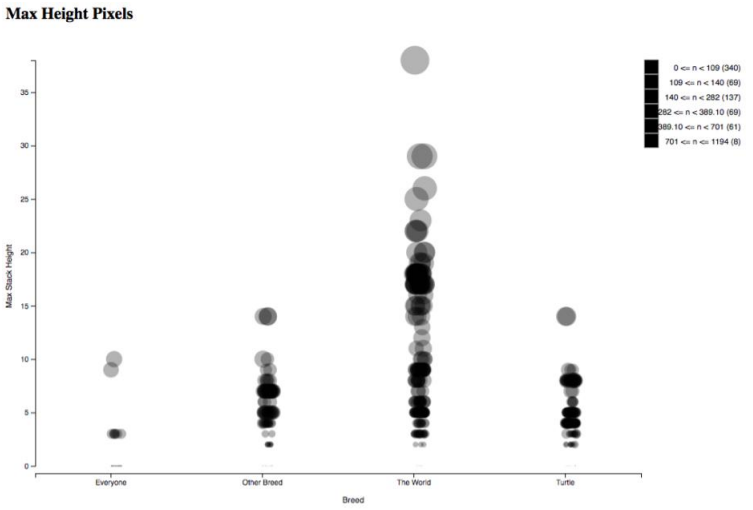
- The World is Complex The blocks in 'The World' breed can become visually overwhelming. The tab layout does not improve this visual block overload much.

5.4 What is the extent of required scrolling on the development page?

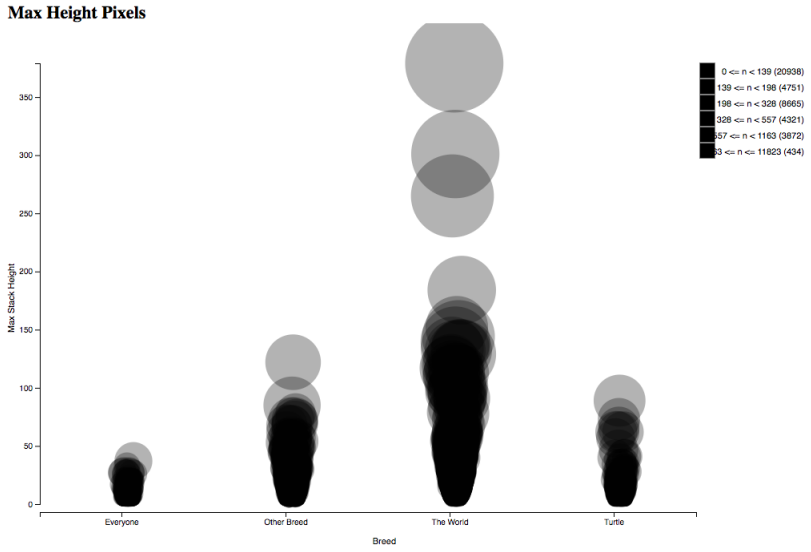
The next few charts visualize how and where users place their blocks on the screen. To better understand the pixel values conveyed in these charts, we will compare the measurements to a standard 1366 by 768 screen resolution computer screen [23]. Overall, the widths, heights, and positions of the stacks in StarLogo projects impact the amount of vertical and horizontal scrolling required for the user.

5.4.1 Vertical Scrolling

As mentioned in the beginning of this thesis, users need to scroll all the way to the bottom of the development page in order to see the blocks in their project. In order to see the program that they have created with their blocks, they would need to scroll back up to Spaceland. This vertical scrolling is enforced by the layout of the development page as a whole. The user would need to scroll even more depending on how their blocks look on the canvas. Consider Figure 5-11, which contains two more circle charts generated by our data analytics tool. The Y-Axis represents the number of blocks in a project and the radius of each circle represents the height in pixels of the tallest stack for a given breed, per project. The maximum height in pixels for Figure 5-10a is 1194 pixels. This means that the tallest stack of the top 200 project signatures is 1194 pixels tall. On a standard 1366 by 768 computer display, a 1194 pixel-tall stack of blocks would require 1.5 scrolls. However, if we take a look at the maximum height pixels of our random sample, we can already tell by how huge the circles are that stacks are substantially larger. In fact, the largest stack in our random sample was 11,823 pixels tall. On our standard computer screen, that would require



(a) Top 200 Project Signatures



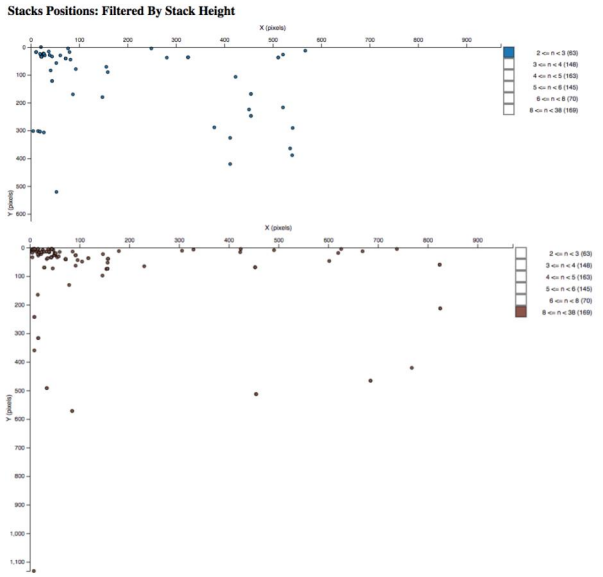
(b) Random 10k Project Signatures

Figure 5-11: Max Height Pixels Across Different Breeds

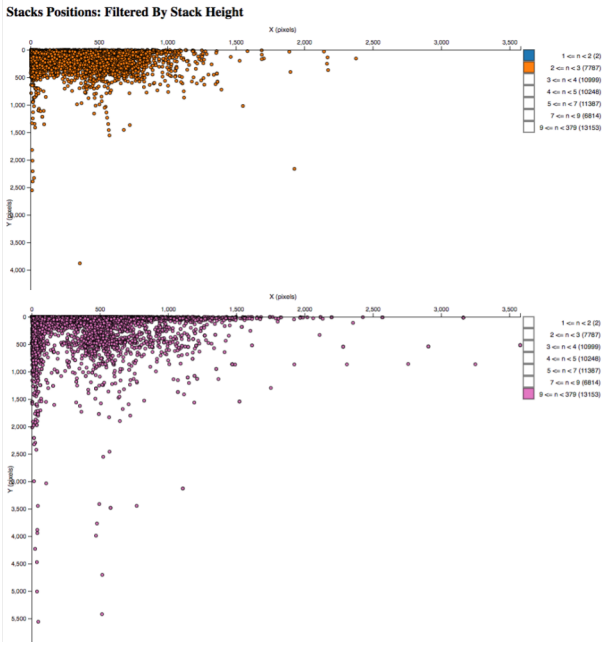
about 15 full screen scrolls in order to look through every block in the stack. If a user edits this tall stack at the bottom and then wants to run their code, they would have to scroll down the development page to the canvas, scroll 15 times to the bottom of the canvas, possibly scroll 15 times back up the canvas to double check their code, and then scroll one last time on the development page to Spaceland. After manually opening these large outlier projects in StarLogo, the project loading performance was also very noticeably slowed. Our data analytics tool reveals the type of usability problems that StarLogo's expert users may run into.

Positioning of block stacks also affects scrolling. Users can place their blocks anywhere on the canvas, expanding the canvas size while doing so. Figure 5-12 gives us a glimpse of this, as it visualizes the stack positions, filterable by stack height. Every point on this scatter plot represents every active stack of blocks that exists in a given sample of projects. The X and Y axes are in pixels, and are positioned in such a way to reflect what the canvas coordinates are on the StarLogo page. Therefore, the placement of the points indicates where the top level blocks are on the canvas in StarLogo. A quick glance at this scatter plot is a bird's eye view of what would happen if every StarLogo Nova project on the website was overlaid on top of each other. This legend is togglable by stack height. We see here that as the stacks get taller, the positions start pushing the canvas limits further, thus expanding the canvas to be taller and wider.

Figure 5-12a represents the positions for all of the active stacks in the top 200 popular project signatures. The top chart in Figure 5-12a displays the positions of the smaller stacks and the bottom displays the positions of the tallest stacks. The lowest y-position on this chart is around 500 pixels for the shortest stacks and is around 1100 for the tallest stacks. On our standard screen resolution of 1366 by 768, this means no required scrolling for shorter stacks, but a small scroll for the tallest stacks. Our random sample that is represented in the charts in Figure 5-12b, however, display a wide variety of usage beyond basic tutorial remixes. The top chart in Figure 5-12b represents the positions of the smaller stacks and the bottom chart in Figure 5-12b represents the positions of the tallest stacks. The lowest y-position

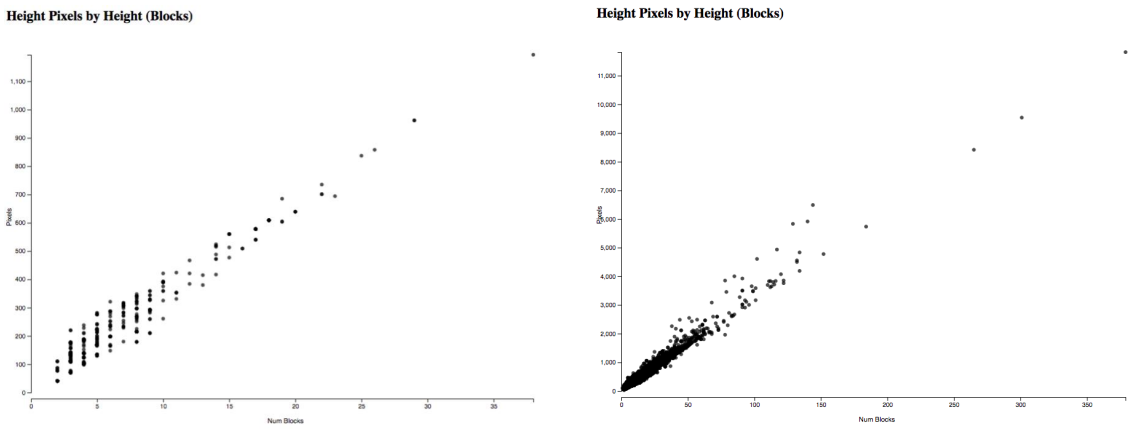


(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

Figure 5-12: Stack Positions Filtered By Stack Height



(a) Top 200 Project Signatures (b) Random 10k Project Signatures

Figure 5-13: Height Pixels by Height Blocks

is around 2500 pixels for the shorter stacks and 5500 for the tallest ones. This would translate to at least 3 scrolls for the shorter stacks and at least 7 full screen scrolls for the tallest stacks. Stack height and stack positions could impact the usability for StarLogo users, both for those who choose to make very tall stacks and for those who just choose to place their blocks on the edges of the screen.

We can additionally take a look at Figure 5-13, which shows the rate of change in the height in pixels as the number of vertically stacked blocks increases. Both Figures 5-13a and 5-13b show a fairly linear relationship. Rough calculations of each charts' slopes yields a value of around 40 pixels/block for the left and 30 pixels/block for the right. This represents the average height of the blocks being used in the top 200 signatures and in our random sample, respectively. The linear relationship indicates that the growth in stack height pixels is caused by an increase in the number of blocks. This trend is not the case for width, as discussed in the next section.

5.4.2 Horizontal Scrolling

StarLogo users may need to do a significant amount of horizontal scrolling on the platform. This is mainly because StarLogo blocks automatically resize width-wise, as discussed earlier in this thesis. Note that the drawer, the container of all of the blocks, takes up about 237 pixels including the left margin- which is about 17% of

the width of a standard 1366 by 768 laptop screen. This is glued to the left side of the screen, so a blank canvas is a maximum of $1366 - 237 = 1129$ pixels.

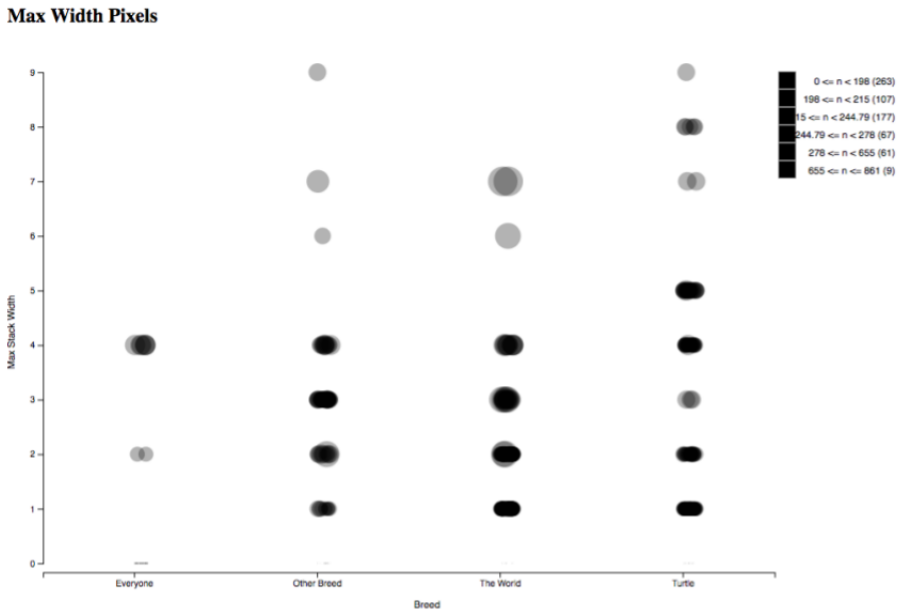
Figure 5-14 is a circle chart similar to Figure 5-11, except that the circle radii refer to the width in pixels of the widest stacks in a given breed for a given project. On Figure 5-14a, the widest stack is 861 pixels, which is less than the 1129 pixel width default of the canvas. We can conclude that these projects' stacks are not very wide and do not require much horizontal scrolling, especially since the top 200 project signatures represent simple tutorial remixes.

However, our random sample includes projects that are more complex than the standard tutorial remixes. The maximum width in pixels for Figure 5-14b is 5416 pixels, which translates to almost 5 large width-wise scrolls, considering the 1129 pixel default width. This is not ideal, since sideways scrolling is not as natural or as widely used of an interaction as vertical scrolling [22].

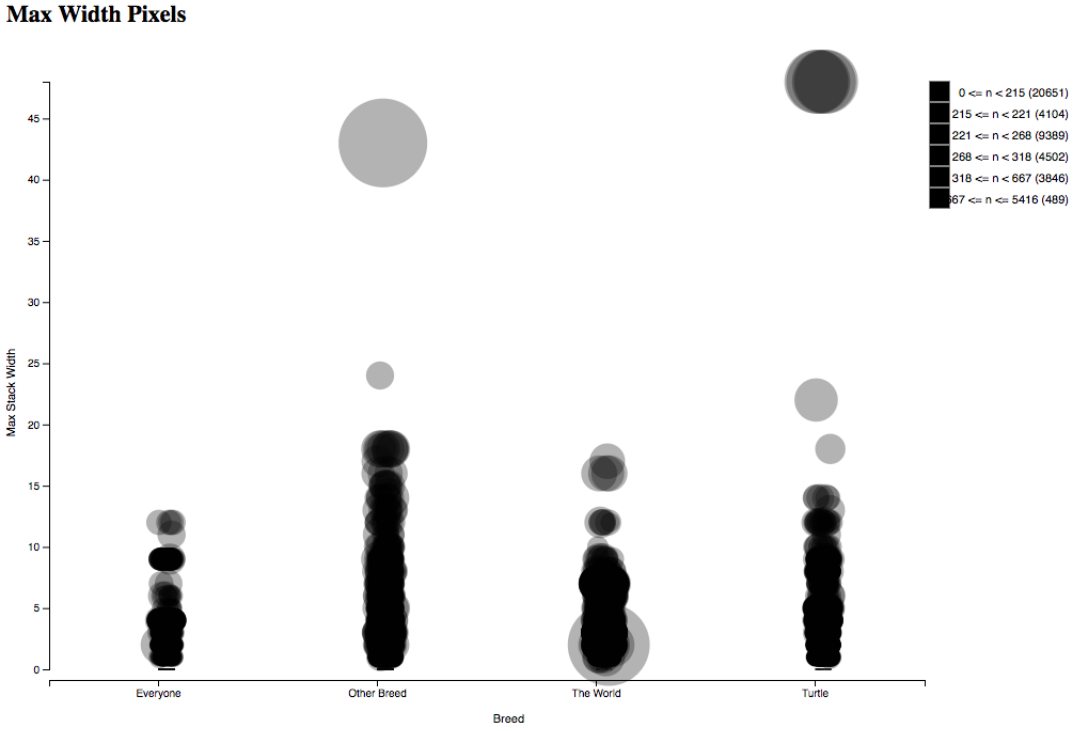
We can look back at the scatter plot in Figure 5-11 too and see that the top 200 project signatures' positions have a maximum x-position of around 600 pixels for small stacks and 900 pixels for taller stacks. This means horizontal scrolling is not necessary. However, if we look at the chart generated by the random sample, the x-position of the stacks in the two charts on the right are 2500 pixels for shorter stacks and 3500 pixels for taller stacks. This translates to around 2 or 3 large width-wise scrolls.

As I opened some projects manually on the StarLogo platform, I realized that I had to manually scroll to the right, even if just a little bit, in order to see that there were stacks of blocks off-screen. This was because, as expected, users are not accustomed to needing to scroll horizontally to reveal new content [22]. Because users can choose to place their blocks anywhere on the screen, they may choose to place them far off to the side or even to the bottom as a way to hide the blocks, and potentially forget about them.

Figure 5-15 is another box-and-whisker plot that shows how the width of a stack in pixels changes by the width in number of blocks. In Figure 5-15a, there does not seem to be much correlation since it is a visualization of the top 200 project

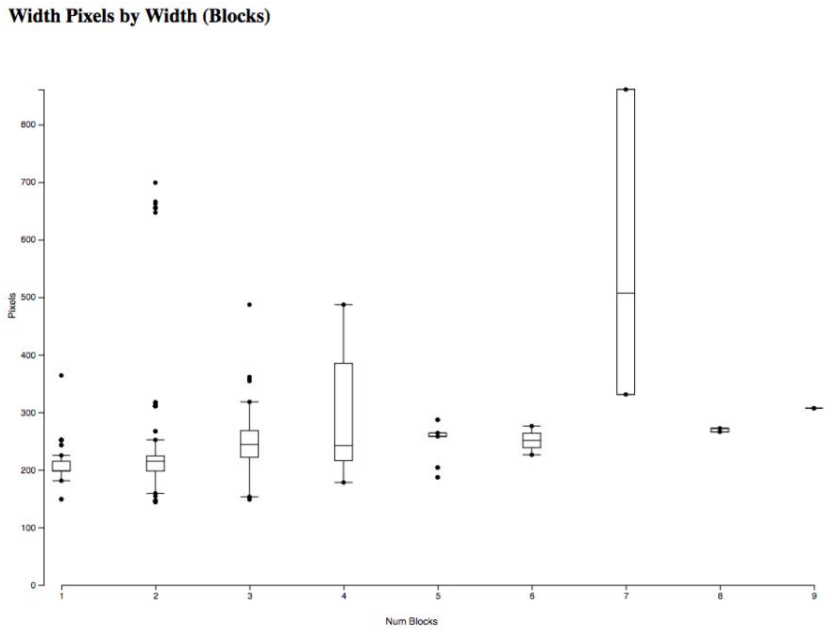


(a) Top 200 Project Signatures

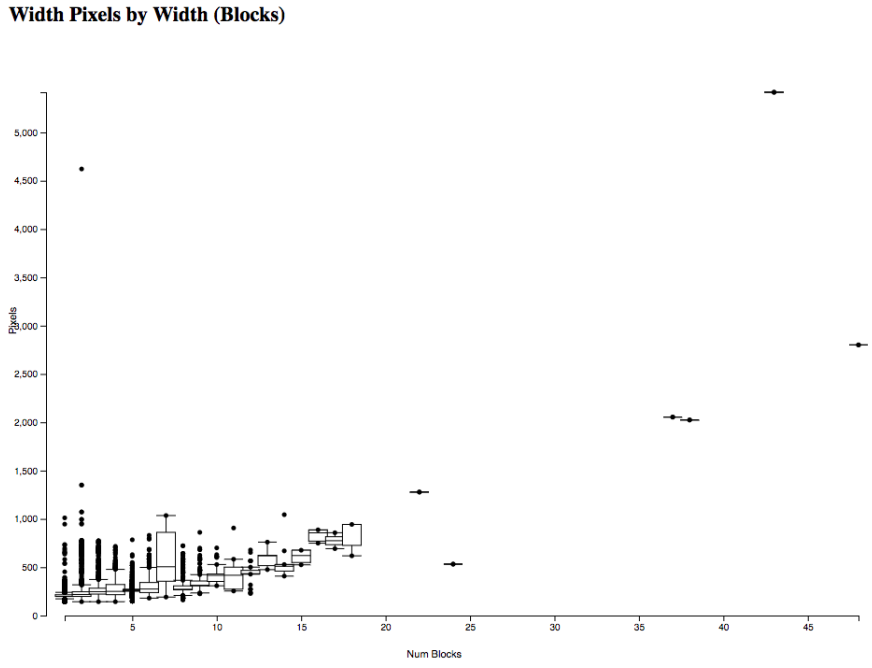


(b) Random 10k Project Signatures

Figure 5-14: Max Width Pixels Across Different Breeds



(a) Top 200 Project Signatures



(b) Random 10k Project Signatures

Figure 5-15: Width Pixels by Width Blocks

signatures, which are simpler projects. However, Figure 5-15b follows an almost exponential trend, which clearly indicates how blocks expand in width very rapidly. Future iterations of this project may involve more exact calculations of the trend line for this particular graph.

5.4.3 Overview: An Overwhelming Amount of Scrolling

We do not know exactly how users feel without conducting user studies, but we have calculated the amount of scrolling that is caused by block sizes and block positions on the canvas.

- Users Must Scroll, Regardless of Expertise The layout of the development page itself requires scrolling down to reach to the canvas and scrolling back up to view Spaceland. Users of any experience level would need to frequently scroll up and down between their blocks and their program. Also, users of any expertise can place their blocks anywhere on the canvas, so users may need to scroll a large amount to find their blocks, if they placed their blocks near the edges of the canvas.

- Vertical Scrolling Aside from the Spaceland to Canvas and Canvas to Spaceland scrolling, there is still a large amount of vertical scrolling required on the platform. Blocks grow tall as users add more children blocks, so users may need to scroll up and down to see all of the blocks in their taller stacks and to add blocks to these stacks. This issue worsens as projects grow more advanced and complex.

- Horizontal Scrolling When blocks get very wide, which happens quickly, users may need to scroll horizontally frequently. This is particularly an issue for projects with more advanced logic, or with projects that use math expression blocks.

5.5 Summary

After observing the visualization results generated by our tool and opening some projects manually, we were able to make the following observations about StarLogo Nova. Some of these attributes were already known by researchers, but the analysis and visualization tool was able to provide more explicit data and holistic backing for the intuitions they had previously.

- A non-trivial number of projects in the database are empty.
- Most popular projects are tutorial-driven, as expected.
- Many projects look like tutorials, even in our random sample.
- The number of total blocks in a project differ from those of the tutorial, but many stay around the popular sizes (9 blocks and 45 blocks). An explanation for this is not yet determined.
- We can assume users generally understand how to use the 'Everyone' and 'The World' breeds.
- Breed tabs can get overwhelming and do not convey information about the project as efficiently as we would like.
- Users, both new and expert ones, are required to scroll vertically and horizontally fairly often.

Chapter 6

Conclusions

In this chapter, we document the impacts that our StarLogo Analytics tool can have on the future of the platform. We discuss user interface recommendations drawn from the visualizations we generated with the tool as well. Limitations and future work are also discussed.

6.1 How Do the Results Address Our Questions?

We have been investigating our data in hopes of finding out ways to raise the ceilings for expert users and lower the floors for beginner users. In particular, we wanted to know how tutorials, breed tabs, scrolling, and block sizes and positions affect usability. The following list summarizes what we have learned about StarLogo project data and what conclusions could be drawn from them.

1. Most Projects are either Functionally Empty or Completely Empty

   This implies that a significant portion of users are confused about how to start a project at all on StarLogo. Rethinking tutorials and how they are introduced to new users may help. In addition, the layout of the development page can be redesigned such that users can see the blocks right away as soon as they land on the development page. Currently, users need to know to scroll all the way to the bottom of the page to start building programs, so theoretically, if users

do not know to scroll down, get confused, and give up, their projects would be Completely Empty.

2. Many Projects are Similar to Tutorials

   At least 24.6% of the projects in the database have signatures that represent some remix of an IPWIT or Project GUTS tutorial, or are functionally empty or close to functionally empty. The breed complexity, stack sizes, and positions of the other 63% of non-empty database projects imply that many of these other projects are built on top of tutorials in some way, too. It is implied, then, that many StarLogo projects are similar to tutorials, are fairly basic, or stem from a tutorial in some way. Therefore, StarLogo could improve in encouraging users to be more creative with the platform beyond the tutorials provided to them. To address this, further investigation and user studies would need to be conducted to see how new users become experts of StarLogo.

3. Breed Tabs Can Get Overwhelming

   Even with more than 10 breeds, the breed tabs themselves create visual overload. Also, it is difficult to know the content of a breed just by looking at the breed tabs. Users must click the tabs in order to see what the tab consists of. Even then, it is not clear to users if there are off-screen blocks that are only viewable if users scroll horizontally. Inefficiencies and visual overload may indicate a need to rethink how breeds are being conveyed on the platform. Also, adding a thicker border of whitespace on the right of the screen could be a stronger indicator to users of whether there are off-screen blocks.

4. Blocks Contribute to Excessive Required Scrolling

   Stacks of blocks can be placed in many positions and can grow, in theory, infinitely tall or wide. The platform requires users to scroll vertically and horizontally in large amounts fairly frequently because of the layout and how the blocks expand. 'The World' breed is particularly complex and disorganized. The visual overload and disorganization of blocks on StarLogo can hinder users' desire

to be creative and to make more complex projects.

In the next section, we discuss suggestions on how to update the StarLogo platform in order to address what we have learned from our results.

6.2 User Interface Suggestions

After analyzing the visualizations generated by our new analytics tool as well as manually opening some outlier projects on the StarLogo platform, we have compiled a list of recommended user interface updates. These suggestions are informed by our data and aim to improve the platform such that floors are lowered and ceilings are raised. Figure 6-1 displays a low fidelity mock-up of a proposed development page redesign.

Side-By-Side Spaceland and Canvas

Being able to see the code while running the code is helpful in debugging. Placing Spaceland and the canvas side-by-side would be useful for both early and advanced end users of StarLogo. It would enable users to see the code behind their simulations and games while they are running the code, thus showing beginner users the bridge between their code and their final program.

In addition, popular block-programming platforms like code.org and Scratch use a similar side-by-side layout as shown in Figure 6-2 [3][9]. Using a format that seems to already be widely used could improve learnability of the StarLogo system. Changing the layout in this way would decrease the overall amount of vertical scrolling required for the user, and expose users to the canvas earlier on the page. As mentioned in the previous section, having the canvas at the bottom of the page could mislead students who do not scroll all the way down to look for it, which could be one contributing factor for the 9.1% of Completely Empty projects in the database. Moving the canvas higher up could counteract that.

The canvas height would be stretched out to be 15% taller than the default canvas height (The minimap can be collapsed, as described in the next two sections). With

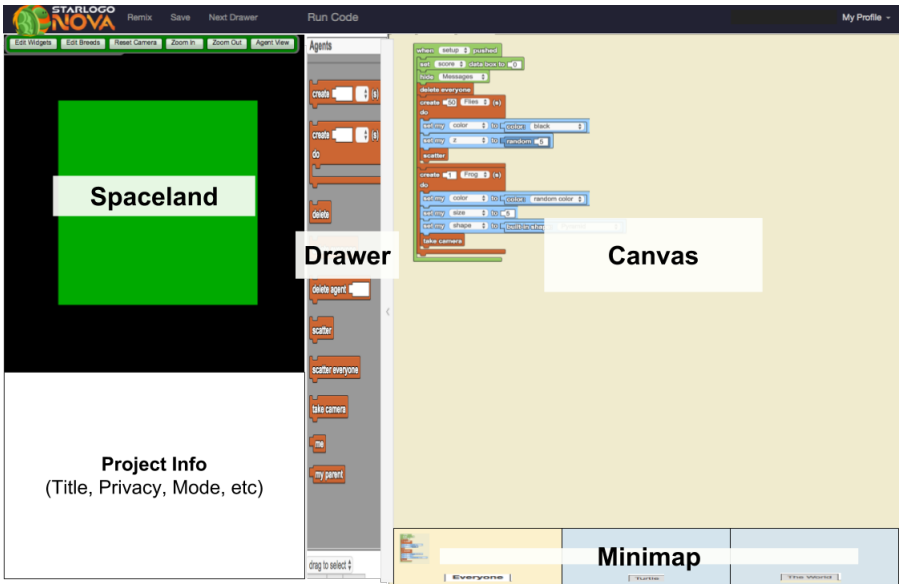


Figure 6-1: Proposed Mock-Up of Development Page Redesign

this new layout, the width of the canvas would decrease by around 20% to 40% depending on whether the block drawer is opened or not. This may increase the amount of horizontal scrolling, but it is possible that this new canvas, which is taller than it is wide, may affect where users place their blocks. For instance, they may stack their blocks more vertically than horizontally. Blocks can still resize to really large widths, so shrinking the width of the canvas may still encourage users to place their wider blocks off-screen (to the right). To truly know if this change would improve or worsen user experience with the platform, this new layout would need experimentation and testing.

Collapsible Elements

From our own experiences, the ability to manipulate editor windows provides useful flexibility for text editor users. So, I suggest having every major element of this screen collapsible to provide greater flexibility for users. For instance, the drawer is about 237 pixels wide, which is a significant amount of screen real estate that could otherwise be used for more blocks or for more of Spaceland. This proposed redesign gives users the flexibility to hide the block drawer, which could be beneficial for all user levels. This suggestion is driven by the other suggestion to move Spaceland and

the canvas to be side by side. A collapsible drawer would be able to bring back more width to the canvas, thus reducing any newly introduced sideways scrolling.

Spaceland can also be minimized, so at best, the canvas can be the full 1366 pixels wide if the user is using a 1366 by 1768 screen. The freedom to expand or hide Spaceland, the canvas, the drawer, the minimap, and the project information box could give users more freedom in where they put their blocks as well as lessen the visual overload that more complex projects possess.

Minimap

No longer being developed, StarLogo TNG was a desktop application that preceded StarLogo Nova. TNG had implemented a minimap which is a translucent display that gives a bird's eye view of the code. Minimaps allow users to see how their blocks look as a whole, which can make navigation between different parts of the page easier. Users could drag blocks between breeds, navigate between different breeds using the map, and get a glimpse of where ones' blocks were placed on any given breed [12].

In Figure 6-1, I recommend the same concept to be added at the bottom of the canvas in StarLogo. The highlighted box in the minimap represents the breed that is currently selected, which would replace the current tab layout of the breeds. This way, users have a preview of which breed they are navigating to without needing to guess the content of each. This would be especially useful for expert users who have more breeds in their projects. Seeing a preview of each breed's canvas means users will know if blocks are hiding in the corners of the screen, too. The minimap would resize when there are more breeds in a project. This is one solution to the overwhelming breed tabs and excessive scrolling issues mentioned earlier.

6.3 Analytics Tool Impacts

Aside from revealing the weaknesses of StarLogo Nova and elucidating information on how users use the system, our analytics tool has other impacts as well. During and before development of our tool, changes were made to the Beta version of StarLogo

Nova that had not been released in the live version yet. Our tool provides data visualizations that support many of these decisions and are relevant to these new changes. We discuss the tool's effect of inspiring new questions among the research team as well.

6.3.1 Verification of Currently Implemented Updates

While this tool was being developed, other StarLogo team members were working on changes to the platform to improve the system. Their tasks were informed by 'good guesses' and observations of the platform, but not from any explicit data results. We discuss one example of a concurrent change that our tool's visualizations corroborate. This example proves that our new analytics tool gives more concrete reasoning for user interface decisions and its results are aligned with researchers' intuitions about the platform.

Widget Bug

StarLogo researchers were aware of a bug on the platform such that when the screen is resized, all the widgets would disappear. This bug was made known to researchers through user feedback, but the extent of the bug's prevalence for users was unclear. The bug, at the time of the writing of this thesis, is fixed, but the measurable impact of the fix was hard to determine. With the help of our data analytics tool, however, we are able to get a better understanding of these statistics. Figure 6-2 is a histogram, generated by our tool, that shows the number of widgets that each project signature's representative project contains. This chart contains the data from our random sample of 10,000 project signatures (the chart for the top 200 project signatures all have 3 or more widgets). As we see in Figure 6-2, 62 of the projects have 0 widgets, which represents 0.62% of project signatures. Thus, the widget bug did not affect too many users, but now researchers better understand the reach of the bug fix. In addition, this type of information can clarify how prevalent a bug is, which can better inform researchers of how to prioritize future bug fixes.

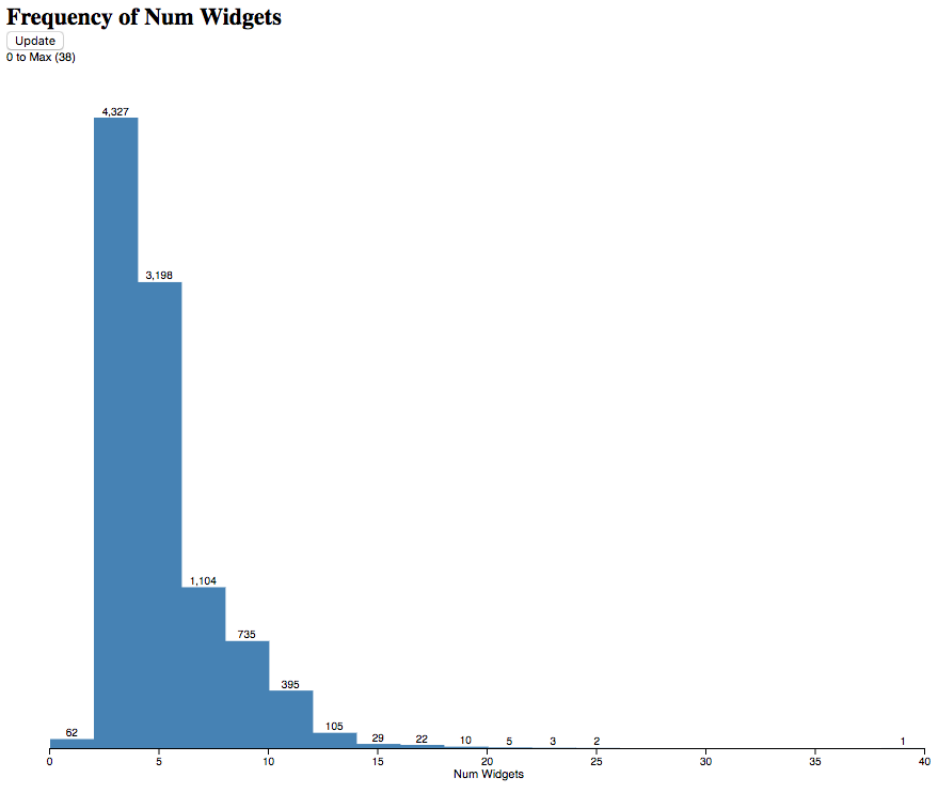


Figure 6-2: Frequency of Number of Widgets

6.3.2 Inspiring New Questions

This preliminary version of our analytics tool sparked intrigue almost immediately when first introduced to the STEP lab. Researchers expressed how they appreciated being able to visually see how StarLogo was being used. One strong indication that this tool is useful is in the number of new questions and requests for different types of visualizations. Researchers were curious to know more, such as how much tutorial remixes differ from tutorials, what the trend lines looked like exactly, and even what users would do if presented with a different visual representation of breeds. Some researchers expressed interest in a similar tool that could be applied to multiple different block-programming languages. This is an indicator of success for our tool, as it provides new information, provides the foundations needed for future user interface experiments, and leads researchers to think of new questions about the platform that had not presented themselves before.

Additionally, we ourselves also asked new questions while developing the tool.

For instance, because some breeds in StarLogo Nova are meant to be empty, it may be useful to redesign how intentionally empty breeds are displayed. Should they be minimized? Should they be placed somewhere else? These are new questions that have been inspired by the visualizations developed by our tool.

6.3.3 Limitations

As mentioned earlier in this thesis, one major limitation of our analysis is that we do not know anything specific about users. This means that we do not know which of the projects visualized in our tool belong to the same users.

It is additionally not clear how many of the projects in the database are actually 'similar', since we filter out functional duplicates and do not measure similarity. By preserving these differences in our analysis, though, we can dive deeper into the question of similarity in future iterations. The trade off was that the analysis we have now does not fully answer the similarity question. We decided to choose the route of creating a solid foundation for future research into similarity. For example, the tree signatures we built, which preserves the differences between similar projects, could be passed into some similarity algorithm in the future. The task of building an accurate similarity algorithm could be an entirely other project in itself.

6.4 Future Work

Our new analytics tool can benefit from automation between extracting and visualizing the data. Adding best fit lines to the charts may also give more concrete numbers to the trends we see. In addition to making the tool generally more usable, the following features can be added as extensions, which would bring more depth to the analysis the visualizations currently provide.

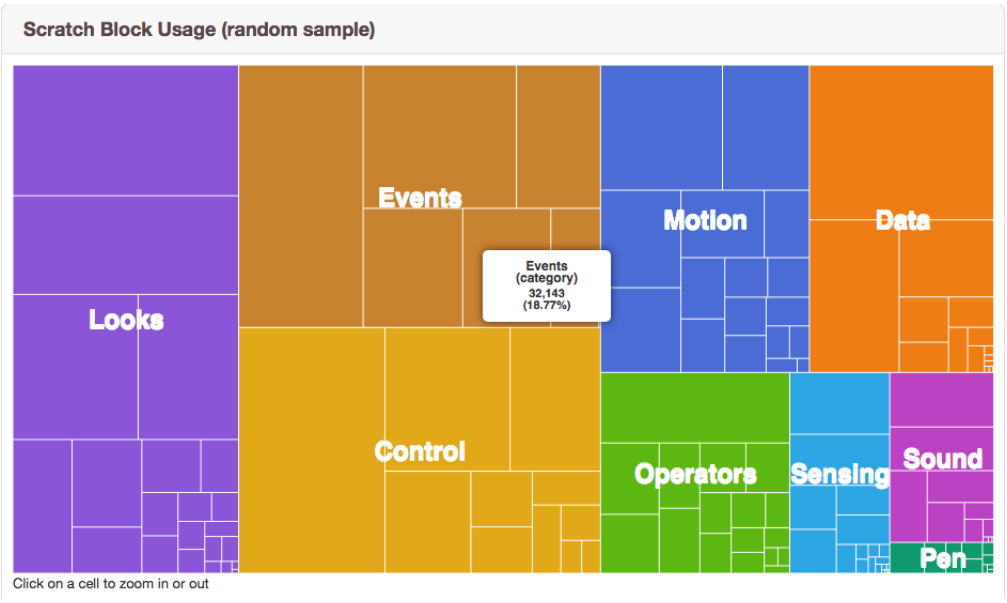


Figure 6-3: ScratchStats' Block Usage Tree Map

6.4.1 Block Usage Tree Map

As discussed in Chapter 2, ScratchStats includes a tree map of the block usage across Scratch. A screenshot of this visualization is included again in Figure 6-3 [10]. Hovering over an individual rectangle displays the percentage of the random sample that has that particular category or type of block. This would be useful to do for StarLogo, as it would inform researchers what blocks are used and what blocks are not. This would thus enable researchers to investigate why certain blocks are used more than others and learn how to improve the blocks that are on the platform. If we use block categories as the grouping criteria, like Xie does in his SuperUROP project, this tree map would also bring us closer to grouping projects by type [25]. Future iterations of our analytics tool could include a tree map like this. The transformed data source generated by our tool includes block names, so this change would simply be extending the code to generate a tree map specific visualization.

6.4.2 Group Projects by Type

Xie's work in App Inventor was able to classify projects by their block usage [15]. StarLogo can benefit from this sort of analysis. Additionally, our analysis tool can

potentially be extended with machine learning techniques to see how much projects cluster toward tutorial projects and toward each other. This would act as a way to measure similarity between projects. Our tree signature string may not work for machine learning methods, since complex string or tree edit-distance algorithms would better fit our string data type. Would identifying each project as a map of $\{blockName : nOccurrences\}$ for all block names work better, as Xie does in his App Inventor analysis [25]? Determining how to represent each project in these machine learning techniques would be an important challenge. Such analyses could help researchers understand how many projects in the entire database are remixes of the Flower Turtle tutorial, or even determine how different projects are from Project GUTS tutorials.

6.4.3 User Analysis

In the future, if researchers are able to gain access to information about individual StarLogo users, more user analysis can be done. This can look like Xie's work, which followed the progress of a user from their first App Inventor project to their later ones [25]. This could also mean creating visualizations of how many users are teachers, of how frequently users use the platform, and of how often users make more than one substantial project. Our existing visualizations could also be updated with this new data by having projects be filterable by certain users, user demographics (i.e. age or locations), or even by user expertise.

6.5 Contributions

In this thesis project, I have contributed the following:

- Gathered and documented pixel width and height data for each block currently in StarLogo Nova
- Built an analytics tool that extracts data and visualizes data about StarLogo Nova's block usage

- Analyzed the preliminary visualizations generated by the tool in order to investigate what the most popular types of projects are
- Used the visualizations to determine how outlier StarLogo projects look
- Provided user interface redesign suggestions for the platform

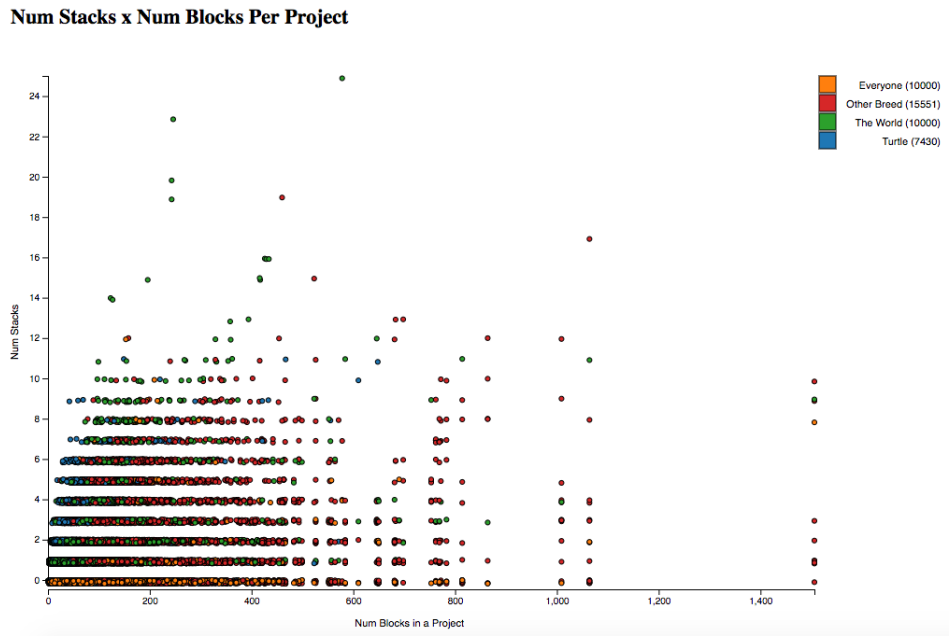
With this new tool, researchers can now extract and visualize their data. With this new ability, they will be able to make more informed decisions regarding redesign of the platform. We hope that this tool is used and extended so that researchers will be able to better understand StarLogo Nova usage and to make more informed decisions on how the platform engages users, how it teaches programming and computational concepts, and how it encourages creativity among users.

Appendix A

Visualizations Generated By Tool

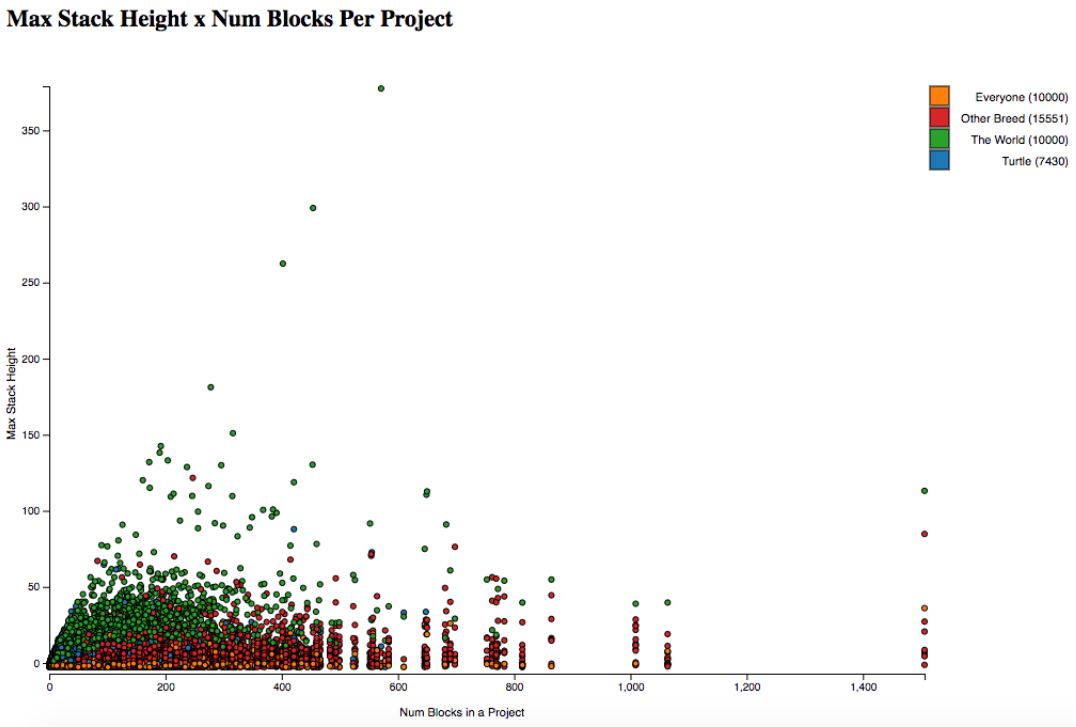
A.0.1 Num Stacks by Num Blocks

The Y-Axis represents the number of stacks and the X-Axis represents the number of total blocks in a project. The legend is togglable by breed. Each point on the chart represents one data point for each breed of every project in the data source. Hovering over a point on the chart reveals a tooltip displaying that point's X and Y coordinates.



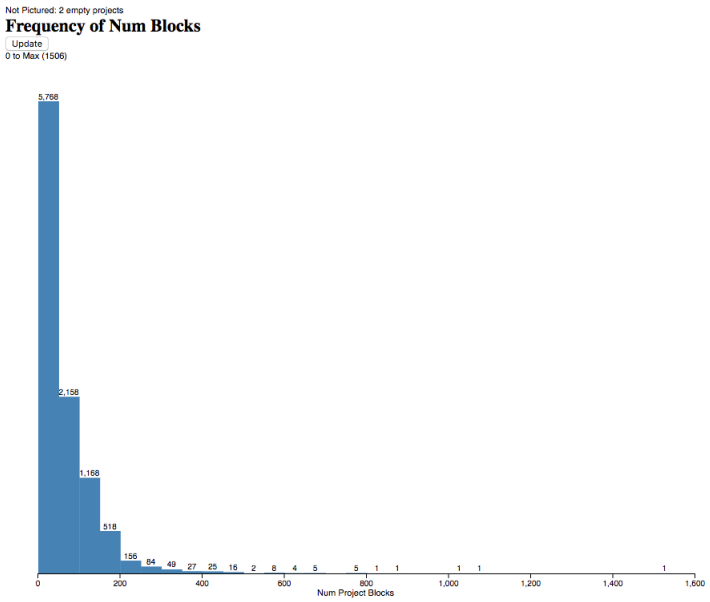
A.0.2 Max Stack Height by Num Blocks

The Y-Axis represents the maximum stack height (in number of blocks) and the X-Axis represents the number of total blocks in a project. The legend is togglable by breed. Each point on the chart represents one data point for each breed of every project in the data source. Hovering over a point on the chart reveals a tooltip displaying that point's X and Y coordinates.



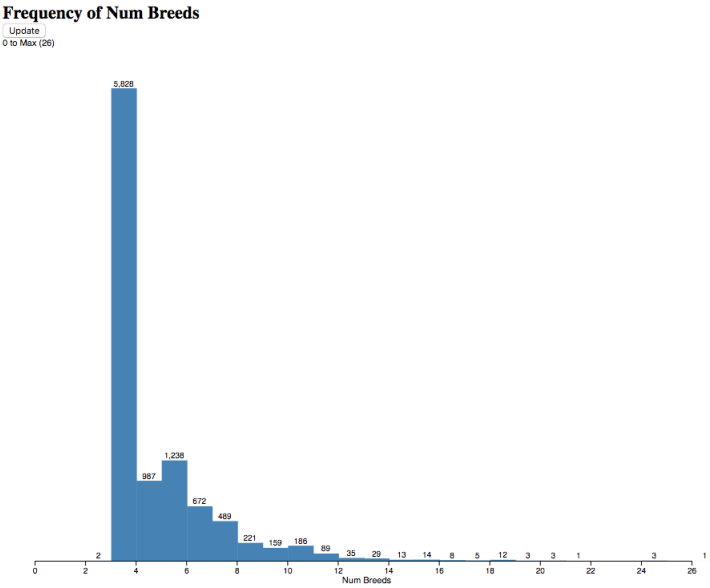
A.0.3 Frequency of Num Blocks

This is a histogram that counts how often each project size occurs in the data source. The X-Axis represents the total number of blocks in a given project. Clicking on the 'Update' button zooms into the histogram, revealing different views between different quartile values.



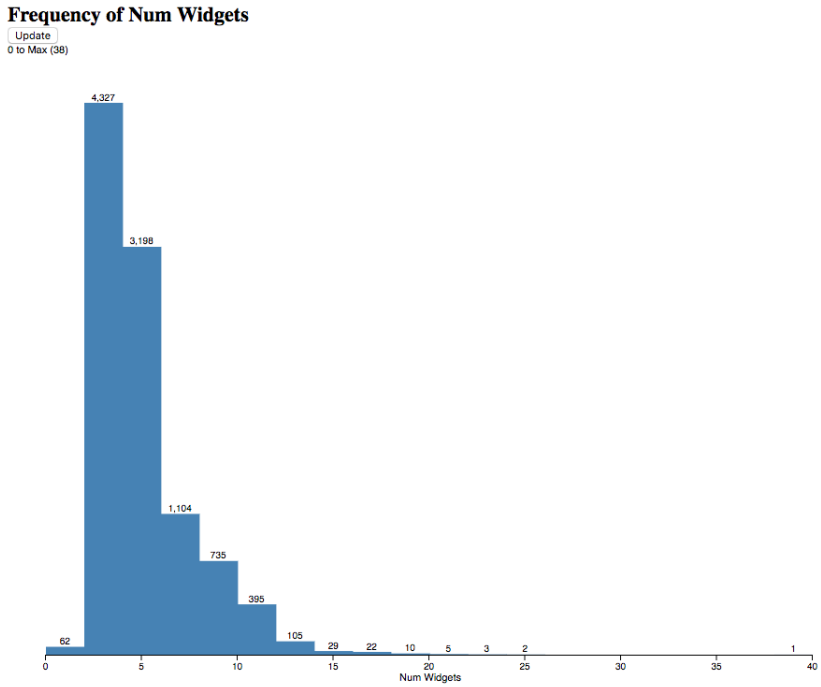
A.0.4 Frequency of Num Breeds

This histogram displays the frequencies of different breed counts in the data source. It shows how many projects in the data source have x number of breeds, where x is some nonnegative integer. The X-Axis represents the total number of breeds that a given project has. Clicking on the 'Update' button zooms the histogram to different quartile views.



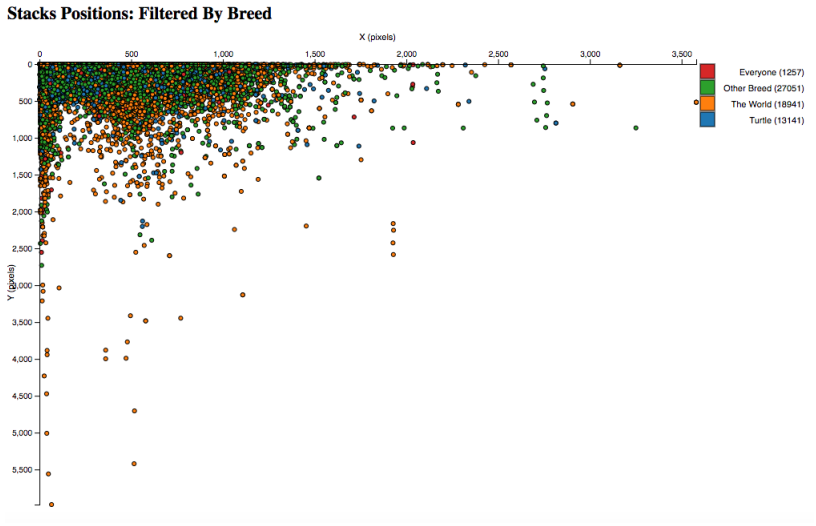
A.0.5 Frequency of Num Widgets

This histogram displays the frequencies of widget counts in the data source. It shows how many projects in the data source have x number of widgets, where x is some nonnegative integer. The X-Axis represents the total number of widgets that a given project has. Clicking on the 'Update' button zooms the histogram to different quartile views.



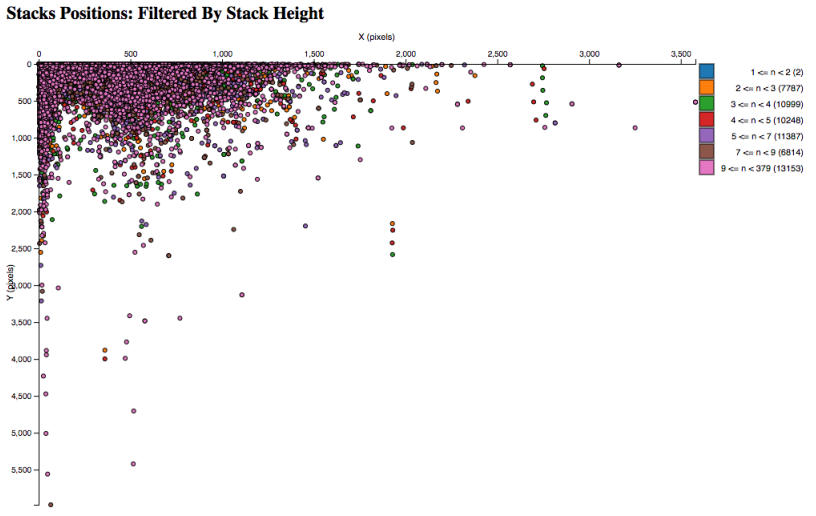
A.0.6 Stack Positions Filtered By Breed

This chart gives a high level depiction of where each active stack in every breed of every project is placed on the canvas. The X-Axis and Y-Axis are in pixels. Each point represents an active stack (i.e. a stack of connected blocks whose top level block is a valid top level block). This chart's legend is togglable by breed. Hovering over a point shows the x- and y-coordinates, the breed name, total number of blocks in the project, project ID, and the top level block name of that stack.



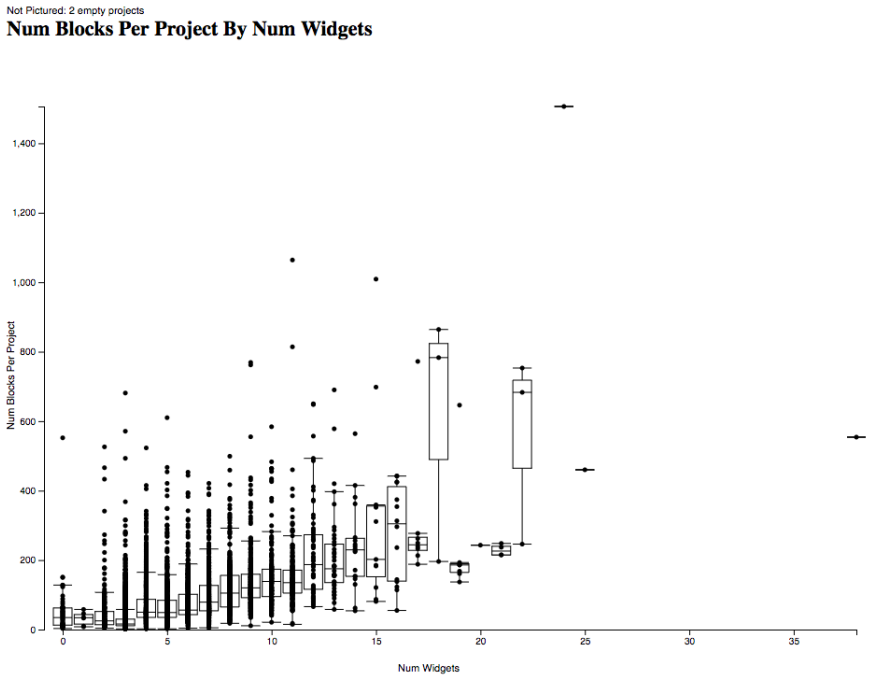
A.0.7 Stack Positions Filtered By Stack Height

This chart gives a high level depiction of where each active stack in every breed of every project is placed on the canvas. The X-Axis and Y-Axis are in pixels. Each point represents an active stack (i.e. a stack of connected blocks whose top level block is a valid top level block). This chart's legend is togglable by stack height (measured in number of blocks). Hovering over a point shows the x- and y-coordinates, the breed name, total number of blocks in the project, project ID, and the top level block name of that stack.



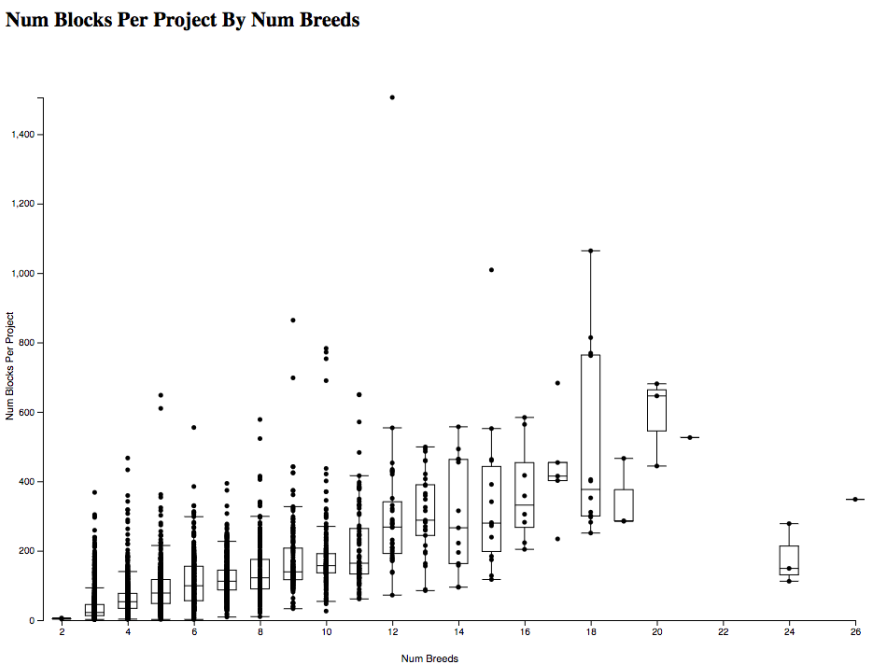
A.0.8 Num Blocks by Num Widgets

This box-and-whisker plot shows what the relationship is between the number of total blocks in a project and the number of widgets that project has. The Y-Axis is the number of blocks in a project and the X-Axis is the number of widgets. Hovering over a box displays a tooltip with the min, max, median, quartile 1, and quartile 3 values. Hovering over a point shows the point's x- and y-coordinates.



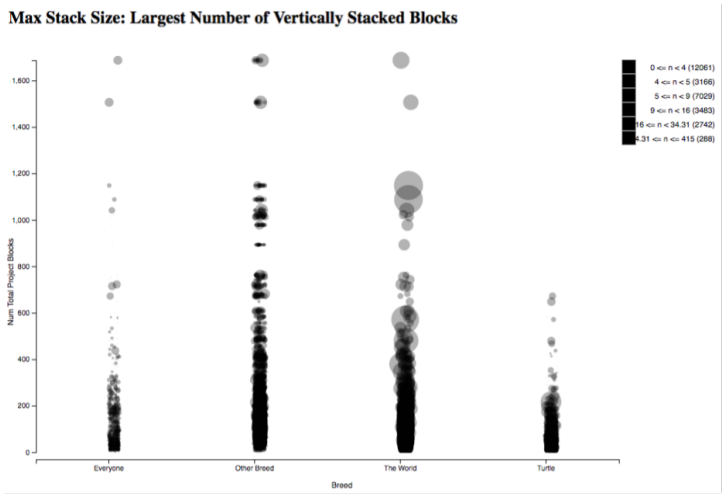
A.0.9 Num Blocks by Num Breeds

This box-and-whisker plot shows what the relationship is between the number of total blocks in a project and the number of breeds that project has. The Y-Axis is the number of blocks in a project and the X-Axis is the number of breeds. Hovering over a box displays a tooltip with the min, max, median, quartile 1, and quartile 3 values. Hovering over a point shows the point's x- and y-coordinates.



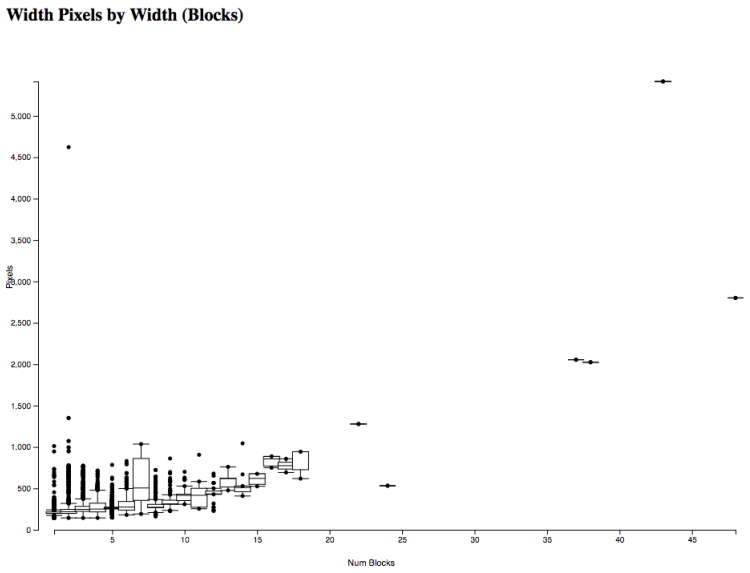
A.0.10 Max Stack Size: Largest Number of Vertically Stacked Blocks

This circle chart compares the maximum stack height, measured in blocks, across different breeds. The Y-Axis is the number of total project blocks. The X-Axis represents the different breeds a project can have in the following order: 'Everyone', 'Other Breed', 'The World', and 'Turtle'. The 'Other Breed' column encompasses the data from all breeds that are not the usual defaults of 'Everyone', 'The World', and 'Turtle'. The legend is togglable by stack height in blocks. The larger the radius, the more blocks contribute to the height of a given stack. Hovering over each point displays a tooltip with project ID, stack height in blocks, and their x- and y-coordinates.



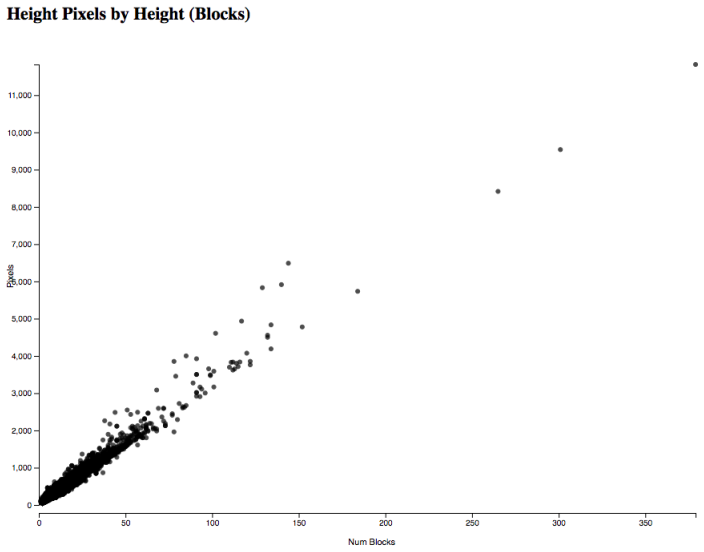
A.0.11 Width Pixels by Width Blocks

This box-and-whisker plot shows what the relationship is between the largest stack width value in pixels and the largest stack width value in number of blocks. This tracks how the number of blocks width-wise relates to how many pixels those blocks are. The Y-Axis is the number of pixels and the X-Axis is the number of blocks that contribute to the maximum width. Each point represents one stack in each breed of every project in the data source. Hovering over a box displays a tooltip with the min, max, median, quartile 1, and quartile 3 values. Hovering over a point shows the point's x- and y-coordinates.



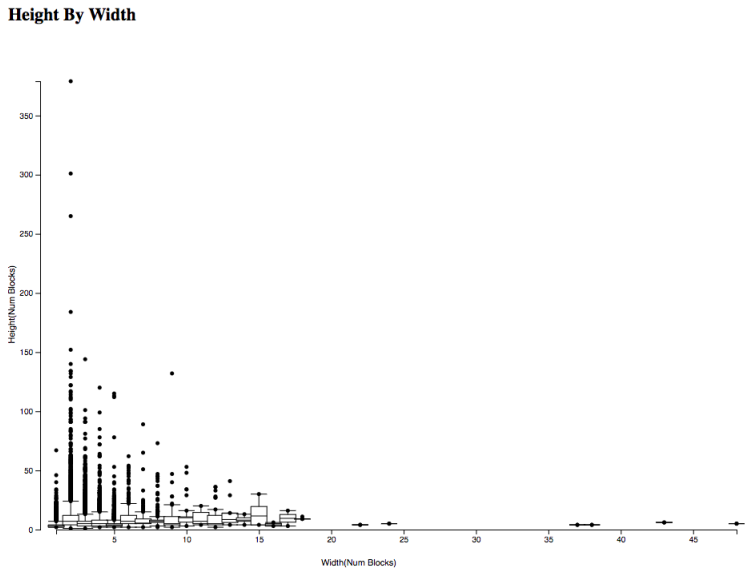
A.0.12 Height Pixels by Height Blocks

The Y-Axis is the height of the tallest stack in pixels and the X-Axis is the height of the tallest stack in number of blocks. Each point represents one stack in each breed of every project in the data source.



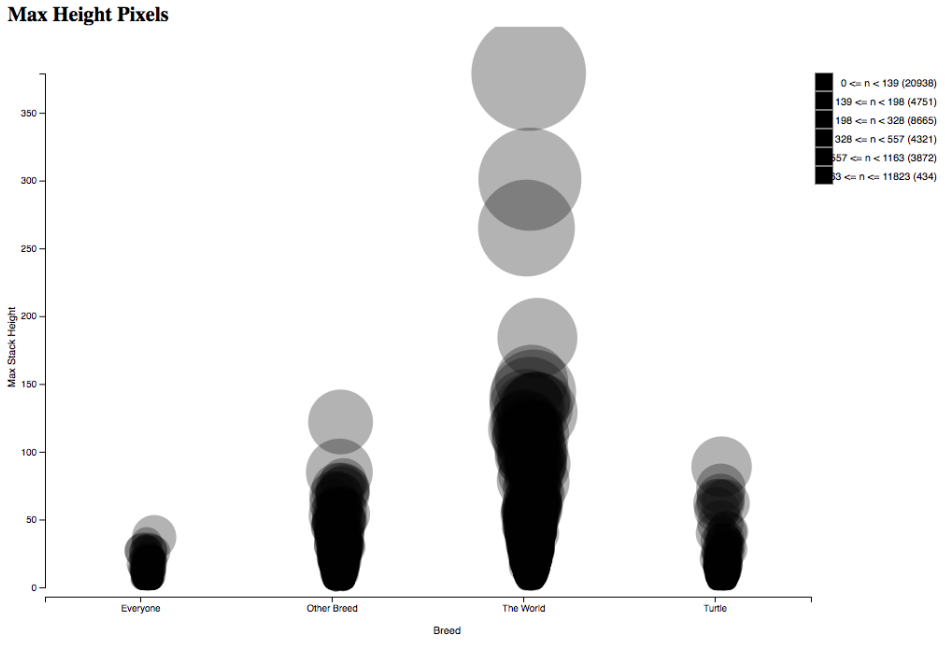
A.0.13 Height Blocks by Width Blocks

This box-and-whisker plot shows what the relationship is between the height of a stack and the width of a stack, both measured in number of blocks. The Y-Axis is the number of blocks height-wise and the X-Axis is the number of blocks width-wise. Each point represents a stack in each breed of every project in the data source. Hovering over a box displays a tooltip with the min, max, median, quartile 1, and quartile 3 values. Hovering over a point shows the point's x- and y-coordinates.



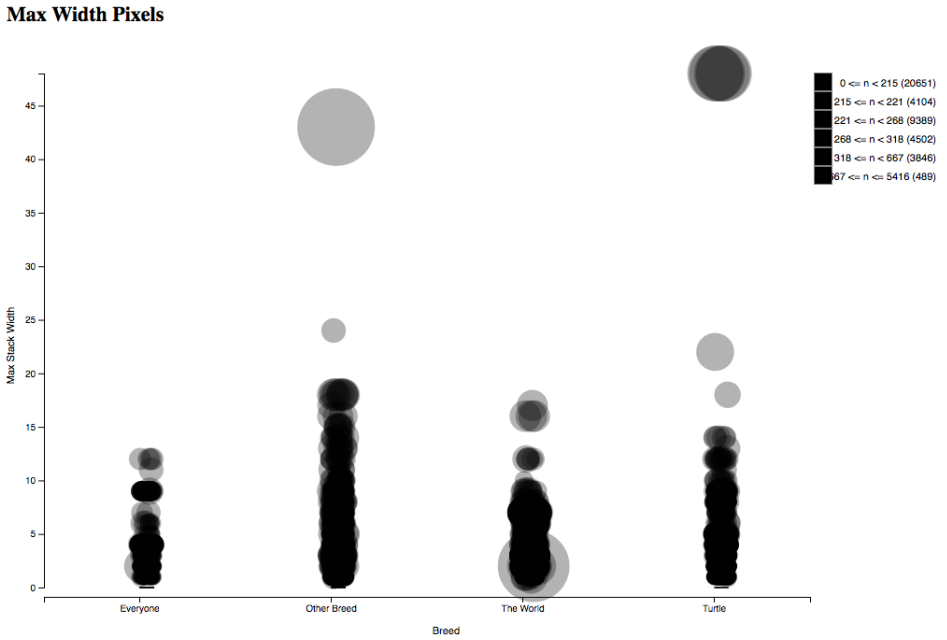
A.0.14 Max Height Pixels

This circle chart shows how tall the tallest stack is in each breed of every project in the data source. The Y-Axis is the number of blocks that contribute to this height value and the X-Axis is the breed name. The legend is togglable by height in pixels. The larger the radius, the larger the number of pixels that contribute to the height of the tallest stack in that project's breed. Hovering over each point displays a tooltip with project ID, stack height in blocks, and their x- and y-coordinates.



A.0.15 Max Width Pixels

This circle chart shows how wide the widest stack is in each breed of every project in the data source. The Y-Axis is the number of blocks that contribute to this width value and the X-Axis is the breed name. The legend is togglable by width in pixels. The larger the radius, the larger the number of pixels that contribute to the width of the widest stack in that project's breed. Hovering over each point displays a tooltip with project ID, stack width in blocks, and their x- and y-coordinates.



Bibliography

[1] Amazon web services. "https://aws.amazon.com/". Accessed July 18, 2017.

[2] App Inventor. "http://appinventor.mit.edu/explore/". Accessed May 30, 2017.

[3] Code.org Studio. "https://studio.code.org/courses". Accessed October 15, 2016.

[4] CS in Science Guide: StarLogo Nova Blocks Introduced. "https://code.org/curriculum/science/files/CS_in_Science_Guides.pdf". Accessed May 10, 2017.

[5] D3: Data Driven Documents. 'https://d3js.org/'. Accessed January 15, 2017.

[6] IPWIT 2013 Activities. "http://www.slnova.org/resources/ipwit-2013-activities/". Accessed February 10, 2016.

[7] MIT Scheller Teacher Education Program Imagination Toolbox. "http://education.mit.edu/portfolio_page/imagination-toolbox/". Accessed June 15, 2017.

[8] Project GUTS: Growing Up Thinking Creatively. "http://www.projectguts.org/". Accessed May 10, 2017.

[9] Scratch. "https://scratch.mit.edu/". Accessed February 20, 2017.

[10] Scratch Statistics. "https://scratch.mit.edu/statistics/". Accessed April 25, 2017.

[11] Starlogo Nova. "http://www.slnova.org/". Accessed June 14, 2017.

[12] Starlogo TNG. "http://education.mit.edu/portfolio_page/starlogo-tng/". Accessed November 10, 2016.

[13] Tableau. "https://www.tableau.com/". Accessed January 15, 2017.

[14] Teachers with GUTS. "https://teacherswithguts.org/members/map". Accessed June 28, 2017.

[15] Xie, Benjamin, Isra Shabir, and Hal Abelson. Measuring the Usability and Capability of App Inventor to Create Mobile Applications. *2015 ACM SIGPLAN Conference on Systems,Programming, Languages and Applications: Software for Humanity (SPLASH)*, October 2015.

[16] Blarg. Page Ruler Chrome Extension. “https://blarg.co.uk/tools/page-ruler”. Accessed January 20, 2017.

[17] Parmenter, Bob, Irene Lee, and Dave Simon. Education & Outreach White Paper: K-12 Educational Program. *New Mexico EPSCoR*, December 2011. “http://archive.nmepscor.org/sites/all/documents/RII4Planning/PARMENTER_whitepaperREVISED.pdf”.

[18] Lee, Irene, Fred Martin, and Katie Apone. Integrating Computational Thinking Across K–8 curriculum. *2014 ACM Inroads Volume 5 Issue 4*, pages 64–71, December 2014.

[19] Lee, Irene, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. Computational Thinking For Youth in Practice. *2011 ACM Inroads Volume 2 Issue 1*, pages 32–37, March 2011.

[20] Odell, James. Obects and Agents Compared. *Journal of Object Technology Volume 1 Number 1*, May 2002. "http://mars.ing.unimo.it/didattica/cas/L4/Odell_ObjectsAgents.pdf".

[21] Pan, Jin. Performance Engineering of the StarLogo Nova Execution Engine. Master's thesis, Massachusetts Institute of Technology, August 2016.

[22] Sherwin, Katie. Beware Horizontal Scrolling and Mimicking Swipe on Desktop. “https://www.nngroup.com/articles/horizontal-scrolling/”. Accessed June 15, 2017.

[23] RapidTables. Screen Resolution Statistics. “http://www.rapidtables.com/web/dev/screen-resolution-statistics.htm”. Accessed July 3, 2017.

[24] Chen, Rita. ScratchStats: A Site for Visualizing and Understanding Scratch Usage Data. Master's thesis, Massachusetts Institute of Technology, June 2010.

[25] Xie, Benjamin Xiang-Yu. Progression of Computational Thinking Skills Demonstrated by App Inventor Users. Master's thesis, Massachusetts Institute of Technology, June 2016.