Usability Improvements for the TaleBlazer Game Editor

by

Cristina Lozano

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2014

Signature redacted

Certified by ....
Professor Eric Klopfer
Director, MIT Teacher Education Program
Thesis Supervisor

Signature redacted

Accepted by ....
Professor Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

Usability Improvements for the TaleBlazer Game Editor

by

Cristina Lozano

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2014, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

TaleBlazer games utilize GPS technology available on most mobile devices and tablets to create location-based augmented reality (AR) games. All TaleBlazer games are created in a web based editor on the TaleBlazer website using a scripts-block programming language. This this describes usability improvements to the TaleBlazer editor that improve the learnability, efficiency, and error correction ability for the game designer. These improvements consist of adding in an agent overview, a printable game summary, an error checker, and a redesign of the styling and layout of the previously existing editor settings.

Thesis Supervisor: Professor Eric Klopfer
Title: Director, MIT Teacher Education Program

Acknowledgments

I'd like to thank Eric Kloper, Lisa Stump, and Judy Perry for welcoming me into the TaleBlazer team and allowing me to work on this project with them for a year and a half. It has been an extremely valuable experience for me and I'm glad to have been a part of such a great project.

I'd like to thank Judy Perry for helping me structure and develop both my UAP and MEng work with her exceedingly valuable project management skills. I would also like to thank Lisa Stump for helping me in the ways of all things TaleBlazer code related and for giving me great guidance throughout the development and implementation of my project.

I would like to thank the entire TaleBlazer Development team, past and present, including my fellow MEngs Tanya Liu, Fidel Sosa, and Stephanie Chang as well as the past MEng students Paul Medlock-Walton and Sarah Lehmann who were invaluable in helping me get started with TaleBlazer. I've shared great times with all of the members of the TaleBlazer team and I especially value their help and feedback from the testing rounds of my project.

I'd also like to thank Jacqueline Hung, my replacement MEng student, for participating in user tests with me and for ensuring that future work will continue on the editor interface redesign.

I would like to thank Juan Rubio of Global Kids and all of his students for allowing me to visit the Brooklyn Public Library and perform user observations and interviews. I would similarly like to thank Susan Baron from the Missouri Botanical Gardens and Cindy Spiva-Evans from the San Diego Zoo for their valuable feedback during user testing.

I'd like to thank my academic supervisor, Eric Grimson whose help and guidance helped me get to where I am today. Finally, I'd like to thank my family, whose moral and financial support have helped me succeed in all of my endeavors at MIT.

Contents

List of Figures

Chapter 1

Introduction

The Scheller Teacher Education Program (STEP) lab has been working on many education games for the purpose of educating children and young adults. The TaleBlazer platform is one of the many research projects that have come out of the STEP lab to take advantage of the burgeoning mobile technology field. TaleBlazer games utilize GPS technology available on most mobile devices and tablets to create location-based augmented reality (AR) games. AR games add digital extras to the world the user is walking through as they reach certain geographic locations. The TaleBlazer platform has an additional component, the web-based game editor, which allows users to create their own augmented reality games with a blocks-based programming language.

1.1 Motivations for Editor Improvements

The TaleBlazer web editor has two target audiences: adult game creators at our partner institutions and groups of middle school or high school aged students at after-school programs or summer camps. Both groups tend to have limited experience with programming and are often minimally familiar with the concept of augmented reality games. We can assume that all of our Editor users are at an age where reading is not a problem but we cannot assume that the user has any knowledge of programming. The editor needs to have a very simple and intuitive interface to minimize the learning curve associated with using TaleBlazer for these user groups. This is a difficult task given the complex nature and customizability of TaleBlazer games. There is a wide range of features that can be implemented with the editor, starting with the basics of game play and ending with some that are very complicated and advanced. Some aspects of these features are very error-prone due to their complex nature. TaleBlazer also has its own vocabulary for many concepts that are unique to TaleBlazer's brand of augmented reality games such as agent, region, player, role, and more that all have their own meaning in the TaleBlazer game world. All of these aspects of the editor make it difficult for a beginner user to quickly and confidently get started making a game.

The original editor presented all of the options to the user without providing guidance as to which features are most crucial and without providing any help dialogues. The user interfaces were not always intuitive for a new user since they required a good grasp on the internal workings of a TaleBlazer game in order to get started. Lastly, they provided the user with a multitude of options without differentiating important, basic features of the game from less important, advanced features.

1.2 Chapter Summary

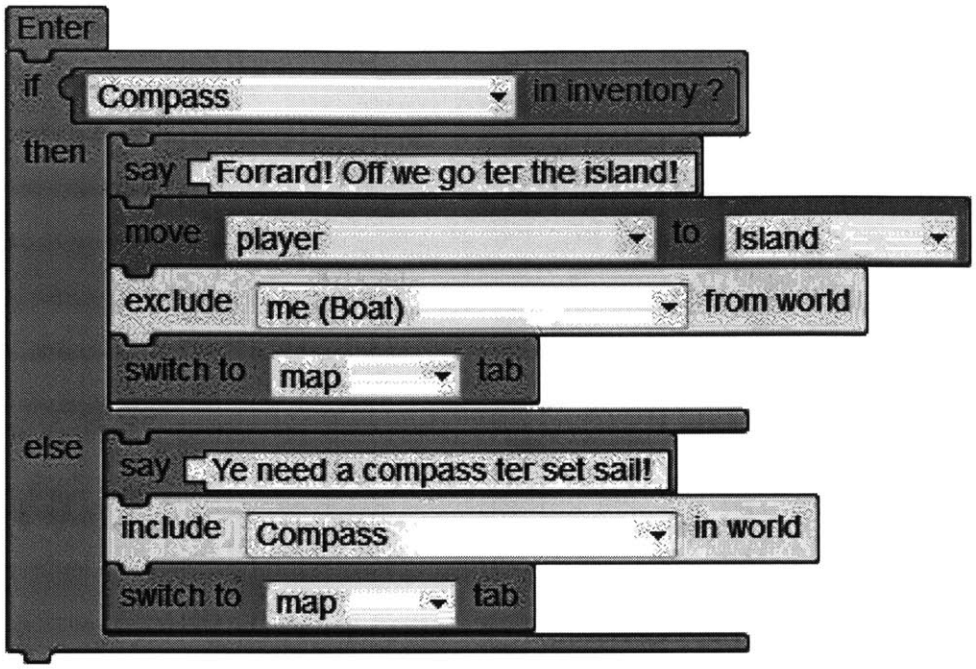
Chapter 2 will explain the background information about TaleBlazer, including previous work that put TaleBlazer where it is now. Chapter 3 will explain the details of the origin of the new usability features as well as the design process for each feature. Since the design of all of the features is an iterative process, chapter 3 outlines the initial design for all of the selected new features. Chapter 4 will detail the testing strategy for every feature and how the design was subsequently altered to respond to user feedback. The final implementation of all of the features is briefly explained in chapter 5. Finally, some possible future improvements to the editor are suggested in chapter 6.

Chapter 2

Background

The TaleBlazer project is a game platform for making and playing location based, educational games. These are called augmented reality (AR) games since they add digital extras to the real world the game player is walking through. Each game is associated with a geo-tagged map that has specific latitude and longitude markers. Characters and objects appear on this map at a latitude and longitude point and have descriptions and often actions associated with them. The player first downloads and installs the TaleBlazer native mobile application and then downloads a TaleBlazer game from the application. The player plays the game by walking around the physical location the game map is representing. The user can interact with characters and objects in the game by first going to their map location as seen by markers on the map and then following dialogues that appear on the mobile device. The mobile device uses GPS coordinates to track where the user is and to tell when they have entered the radius of any marker on the map.

TaleBlazer is currently used by our partners at the Columbus Zoo and Aquarium, Old Sturbridge Village, the San Diego Zoo, the Missouri Botanical Garden, Red Butte Garden, and Drumlin Farm as well as in educational programs such as the Global Kids after-school program based at various locations in New York City.



```
Enter
if Compass in inventory ?
then
    say Forrard! Off we go ter the island!
    move player to Island
    exclude me (Boat) from world
    switch to map tab
else
    say Ye need a compass ter set sail!
    include Compass in world
    switch to map tab
```

Figure 2-1: Example of TaleBlazer Scripts-Blocks

2.0.1 Editor

Each TaleBlazer game is built on TaleBlazer's web-based editor. Each user can see all of their own games on the TaleBlazer website. With the TaleBlazer editor, the user can create a set of regions, which have a map picture and geographic coordinates and the user can place agents that they have created throughout these regions. The user can also make agents, which are essentially any sort of game object with which the player can interact. The game designer can also specify different player roles. The agents and roles can have many traits, actions, and scripts associated with them. The game designer uses a blocks-based programming language on the editor to make scripts that code the game mechanics. This blocks-based language allows the user to drag and drop logic blocks rather than coding in a traditional ASCII programming language to specify the game dynamics. By connecting the blocks in various ways, the game designer can control placement and inclusion of in-game objects and customize the game player's interactions with those objects and locations.

For example, in Figure 2-1, when the 'Enter' action is pressed, if the compass agent is in the player's inventory then a pop-up on the mobile screen will say 'Forrard! Off we go ter the island!', the player will be moved to the Island region, the boat agent will be removed, and the mobile screen will subsequently switch to the Map tab. If the compass is not in the inventory, then the mobile screen will say 'Ye need a compass ter set sail!', the compass agent will be included in the world and the mobile screen will also subsequently switch to the Map tab.

2.0.2 Server

All of the TaleBlazer games are stored on a server that can communicate with mobile devices loaded with the TaleBlazer mobile application. The server also stores all of the assets for every game, such as images and videos. The TaleBlazer editor is implemented in JavaScript on the frontend and has little interaction with the server.

2.0.3 Mobile

The TaleBlazer mobile application is built using Appcelerator's Titanium toolkit, which uses JavaScript and compiles those scripts into native applications for both Android and iOS. The user created game files with all of the compiled game scripts are downloaded to the mobile application when the user downloads any TaleBlazer game from a game code. The mobile application uses the phone's GPS to locate the user on the game map and pings the Server for updated versions of the game.

2.1 Previous Work

The STEP lab and the Media Lab at MIT have worked on many different blocks-based programming systems that have influenced the work on the TaleBlazer platform (see Scratch, AppInventor and StarLogo as examples). There have also been prior attempts to create augmented reality games in the STEP lab as well as the computer science community as a whole.

2.1.1 MITAR Games

Prior to launching TaleBlazer, a previous version of the STEP lab's AR software called MITAR was used. This project was a collaboration with the Education Arcade. While the MITAR platform had similar educational goals through the use of location-based games, MITAR games were less intricate than the current TaleBlazer implementation (e.g., there was no scripting involved) and the games were created via point-and-click filling in numerous forms and templates rather than a blocks-based language. This system was much less usable by children and young adults due to the overhead required to learn the intricacies of the software. A version of the editor called GameBuilder was created with a more limited scope in order to make the process of creating games for learnable. GameBuilder did succeed in being easier for beginners but is severely limited what the game creators could build in a game. [1]

2.1.2 StarLogo

StarLogo is an ongoing project from the STEP lab that also incorporates block based programming into a system to create simulations for modeling decentralized systems. The block based language from StarLogo was adapted to work with the TaleBlazer system.

StarLogo and StarLogo TNG are both pieces of software that require installation, unlike the TaleBlazer editor, which is web, based. StarLogo TNG like TaleBlazer also has many blocks with platform specific terms that are not explained clearly directly in the editor. StarLogo TNG has a similar organization structure for its blocks, but it has even more types of blocks than the TaleBlazer editor, which causes it to be even less learnable to a new user. One benefit of the blocks in StarLogo TNG is the clicking sound that is produced when two blocks are snapped together. One common problem in the TaleBlazer editor is that users often mistakenly think that blocks are connected when in reality they are just placed very closely. The sound effect used in StarLogo is one way to mitigate errors pertaining to block connectivity. [2]

The most recent incarnation of StarLogo is Starlogo Nova, which allows users to create simulations and games in the web browser just as the TaleBlazer editor does. This version of StarLogo is in an earlier stage of development than the TaleBlazer editor. [3]

2.1.3 Scratch

Scratch is another block based programming language created by the Lifelong Kindergarten (LLK) group at the MIT Media Lab. This system is intended to help children learn the basics of programming by creating simple animations and games using drag and drop block programming. The Scratch game editor has a similar interface to the TaleBlazer editor but it includes helpful inline tutorials clearly presented which aid the usability of the application. The Scratch games have different blocks than the TaleBlazer system since the games are not are not location-based. Finally, the Scratch games do not need to account for mobile device settings since the games are all run from the browser. Scratch is similar enough to TaleBlazer that users who have had prior experience using Scratch oftentimes find it easier to get started using TaleBlazer. [4]

Chapter 3

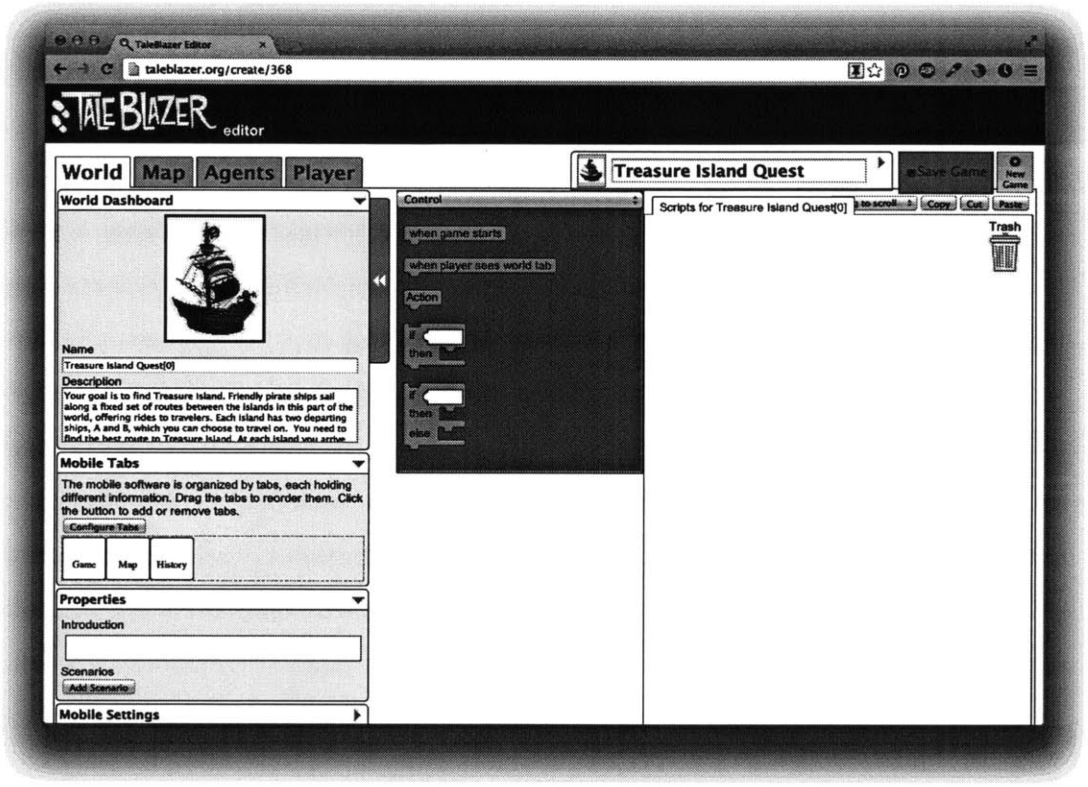
TaleBlazer Editor Initial Design

The TaleBlazer editor is a very powerful game design tool with many complex features. While the feature set is highly developed and intricate, there was much room for improvement for the interface with the user. By increasing the usability of the editor, the strength and the multitude of options present in the editor become more apparent to the average user. The guiding usability principles that all of these designs attempt to improve are the learnability, the efficiency, as well as the ability to correct errors. This section details the design for all of the new features added to the TaleBlazer editor. All of these proposed improvements to the TaleBlazer editor were designed using an iterative user-centered design and testing strategy as described in Chapter 4.

3.1 Original Implementation

The original editor has four tabbed panels that show all of the options for creating a TaleBlazer game from start to finish. The four tabs are World, Map, Agents, and Player in that order; each tab contains pertinent settings for editing that aspect of the game. The World, Agents, and Player tabs are divided into a left panel that has subpanels controlling different properties and a right panel that has the script editor for the world or agent or player. (See Figure 3-1)

The first tab, the World tab , is where all of the settings for the game world



Name: Treasure Island Quest[0]

Description: Your goal is to find Treasure Island. Friendly pirate ships sail along a fixed set of routes between the islands in this part of the world, offering rides to travelers. Each island has two departing ships, A and B, which you can choose to travel on. You need to find the best route to Treasure Island. At each island you arrive

The mobile software is organized by tabs, each holding different information. Drag the tabs to reorder them. Click the button to add or remove tabs.

Figure 3-1: The orginal design of the World tab for the TaleBlazer game editor. The yellow pane (left) is the properties pane and the gray box (center) and adjacent white box (right) is the script editor pane

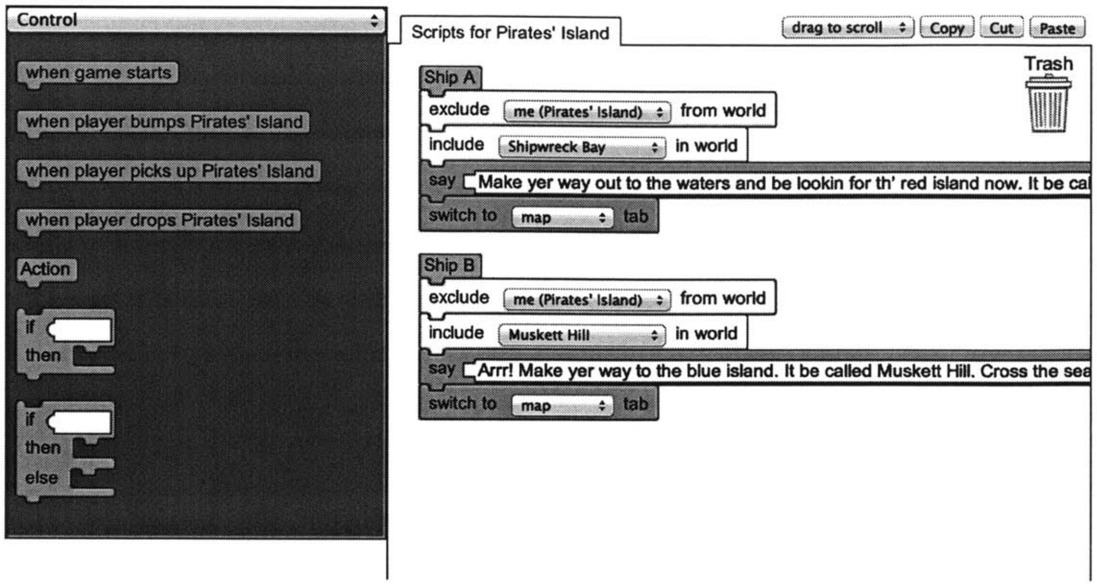
are specified. This includes settings for the mobile user interface, such as which tabs show up on the mobile device, as well as which soft keyboards are presented for typing codes and passwords. The World tab also includes scripts, actions, or traits that are pertinent to the whole game world.

The second tab is the Map tab where all the regions are specified for the game. Each region has a geographic location that can be specified in this tab by GPS coordinates. Each region also has an image for the map and all of the agents in the game show up in their assigned positions on this map image.

The third tab is the Agent tab where the user can see all of their agents as well as create new agents. The view for the Agent tab has a ribbon across the top, similar to other editors such as Microsoft Word, with little tiles for each agent. One agent tile from the ribbon can be selected and the main view shows the settings for the selected agent such as the name, description, password, location, accessibility, and inventory settings. The view also shows all the traits, actions, and scripts for the agent.

The final tab, the Player tab, specifies all the roles in the game. Similar to the Agent tab, the Player tab has a ribbon with all of the roles with one role selected at a time. The name, description, actions, traits, and scripts for the selected role are displayed in the main panel.

Many of the subpanels for properties in the editor were originally laid out with an attempt to follow "What you see is what you get" (WYSIWYG), meaning that the form fields appear in a way similar to how their content would look on the mobile phone. Unfortunately there are enough dissimilarities in appearance that make this aid a little unclear to a novice user and there is no explicit statement of purpose explaining this feature to the user. There are also many unfamiliar terms that are important in the TaleBlazer world, such as agent, player, role, world, etc. that are used heavily in the editor. While some of the more confusing settings have some description of what they do, they are often defined heavily in terms of other TaleBlazer keywords that might likewise be unfamiliar to a new user. There are no inline help pop-ups or fields. On a separate part of the TaleBlazer website there are a series of tutorials to get you started using TaleBlazer, but they are not readily available


Control
when game starts
when player bumps Pirates' Island
when player picks up Pirates' Island
when player drops Pirates' Island
Action
if then
if then else
Scripts for Pirates' Island
drag to scroll
Copy
Cut
Paste
Trash

Ship A
exclude me (Pirates' Island) from world
include Shipwreck Bay in world
say Make yer way out to the waters and be lookin for th' red island now. It be ca
switch to map tab

Ship B
exclude me (Pirates' Island) from world
include Muskett Hill in world
say Arrr! Make yer way to the blue island. It be called Muskett Hill. Cross the se
switch to map tab

Figure 3-2: The orginal design of the editor script panel with the options of blocks of the left and the game designer's scripts on the right

directly from the editor.

The script editor present in the World, Agent, and Player tabs uses the Script-Blocks library and has a panel of available blocks from which the game designer can create their game. These blocks are organized into six categories: control, operators, game, looks, movement, and traits. The user switches between categories by using a dropdown at the top of the available blocks panel. (See Figure 3-2)

One drawback of this system is that games get complicated very quickly and this editor does not have the classic features of a code editor, such as search and replace, refactoring tools, automatic tab formatting, and other tools critical for readability and organization of large sections of code. There also aren't any inline documentation pop-ups or compile errors that editors like Eclipse would give you for a compiled programming language like Java. There is a dearth of error checking and correction tools that flag blocks that are accidentally disconnected or blocks that are unreachable due to faulty logic blocks or script blocks with no script actions attached. Another common problem is that users can delete objects and still have blocks in their scripts that refer to these objects. Finally, there are many blocks that require user input in text or block form but there on no validations to check if these fields are actually populated. All of these common problems cause the games to have runtime errors and leave the user frustrated with no course but to manually search through all of their scripts to find breaking points.

As the games in the TaleBlazer start having more components such as roles and agents and scenarios, the complexity of the games drastically increases. There aren't any tools in the current Editor implementation to get a full overview or summary of the game as a whole, which makes the "big picture" of a TaleBlazer game hard to see amongst all of the details. Without a big picture overview of the components of their game, it is hard for the user to see possible flaws in their design or logic or for them to share their game easily with other users.

3.2 User Research

The TaleBlazer editor is a very powerful tool but there is room for improvement to the functionality and usability. Experienced adult users of the editor filled out a poll sharing feedback regarding possible deficits of the current software and suggestions for new features and improvements. This gave us a sense of what additions or changes would be most useful to that population. Usability concerns for new users were ascertained by observing and then interviewing high school students using the software during an afterschool program conducted byGlobal Kids in Brooklyn, NY.

3.2.1 User Profiles

The current users of TaleBlazer fall into two main categories: young adults learning to create games in school or summer camp programs and adults who create games for our partner organizations. The two target user audiences have very different needs. The learnability aspect of usability is very important for the young adults and children using the product. If the learnability of TaleBlazer increases, the teachers using TaleBlazer will have a much easier time getting their students to pick up and engage with the software. For the adult users who are already accustomed to TaleBlazer and use it regularly to create and maintain games, the efficiency aspect of the usability is key. Both types of users benefit from the system having better error checks and generally being less error prone.

3.2.2 Polling Users

The poll to assess priorities for our experienced audience included options describing improvements to asset management, additional sounds, error checking, content management, and in-editor help. The poll was administered to the principal game creators at the Columbus Zoo and Aquarium, Drumlin Farm, and the Missouri Botanical Gardens as well as the head of the Global Kids educational program. These game creators are all non-technical staff at these institutions who create games for their institutions on TaleBlazer and who teach the TaleBlazer software to camp and after school groups of young adults. These users are all very familiar with the current TaleBlazer software and represented the views of adult game creators, seeking improvements to the efficiency and safety of the interface. The director of the Global Kids program, also had concerns about the learnability of the software since his program shows the software to many new students and after-school facilitators. (See Appendix A for poll)

After pooling the results, the most requested new features were:

1. The ability to generate a printable overview of the game content

2. The inclusion of an overview of the all of the agents in the game

3. An error checker

4. A spell checker

5. Customizable icons for action buttons

The first four of these requests show a desire to have ways to sort through all of the information and content present in most developed TaleBlazer games, either to get a broader view or to isolate any problems.

3.2.3 Global Kids Visit

In October of 2013, the students in a Global Kids afterschool program at the New York Public Library in Brooklyn participated in a user study for the TaleBlazer editor software. The students were all high school age and had limited familiarity with programming prior to starting the afterschool program. Five students sat for a individual interview of about thirty minutes each where they discussed their background in programming, what their experience using TaleBlazer had been like, and what they felt they had learned from using TaleBlazer.

These student interviews offered insights into students' perspectives about the learnability of the existing TaleBlazer editor. The students who had no prior experience with programming all thought that the software was pretty easy to learn, but they also said they felt like the skills they were learning through TaleBlazer were not applicable to "real" programming since it did not have the same syntactical structure they had seen in examples of code. The two students who did have prior experience with programming both said the software was pretty easy to learn but that they would assume it would be hard to learn if you had no experience with programming logic and structure. They felt TaleBlazer was a good introduction to the thought process required for programming and that it made programming more accessible due to the block language's ability to remove questions over syntax. Overall, these interviews suggested that the students were gaining some exposure to basic programming skills whether or not they were aware of it and that the basics of the TaleBlazer software could be learned by young adults over the course of a couple of weeks.

After the interviews, the students worked on building a game they had been designing over the course of several weeks with the guidance a staff member from Global Kids. From observations of their work, it became clear that the students were most confused and least efficient when they needed to flip through multiple panels of content in search of a script or a setting they knew they needed to edit or fix. They also spent more time than necessary trying to batch-edit agents by flipping between agent tabs. For example, one student decided he wanted to change the naming scheme for several agents and he had to flip click through multiple agent tabs changing the name for each. This process is particularly slow since every time a new agent is clicked on, all of the scripts for the agent need to be loaded, which can often be time consuming in terms of loading time if the agent has many associated scripts. This behavior could be helped by adding in the agent overview or game summary that was requested by our partners in the poll.

3.3 Agent Overview Initial Design

The first new feature for the TaleBlazer editor was built in response to observing the Global Kids students struggling with batch updating the TaleBlazer agents and in response to the poll results. The overview is designed to help minimize the number of steps the user has to take in order to change some of the most commonly updated agent fields. A typical TaleBlazer games include a set of agents that appear in locations around the map. Most TaleBlazer games have a large number of agents often placed in multiple regions since an agent very broadly represents any object in the game that isn't the game player. The new agent overview mode gives the game designer a better sense of the agents and their relation to one another. This agent overview is an alternate view to the original properties and scripts pane view with the goal of increasing the efficiency of updating agents and creating new agents for more advanced users. The overview has a tile for each agent that has inline editable fields for the most important characteristics of the agents (e.g., name, description). This new view allows the user to easily scroll vertically through all of the agents and see approximately five or six agents at any given time on the screen (vs. one at a time in the standard view). The agents are also sortable by region, by name, by icon shape, by icon color, and by the custom ordering the user provides.

Before implementing anything in code, there were several iterations of wireframe mockups that helped determine the final layout for the agent overview as it now exists. The starting mockup design of this view has editable form fields for the most critical properties of an agent, which were defined as their name, description, icon, region,

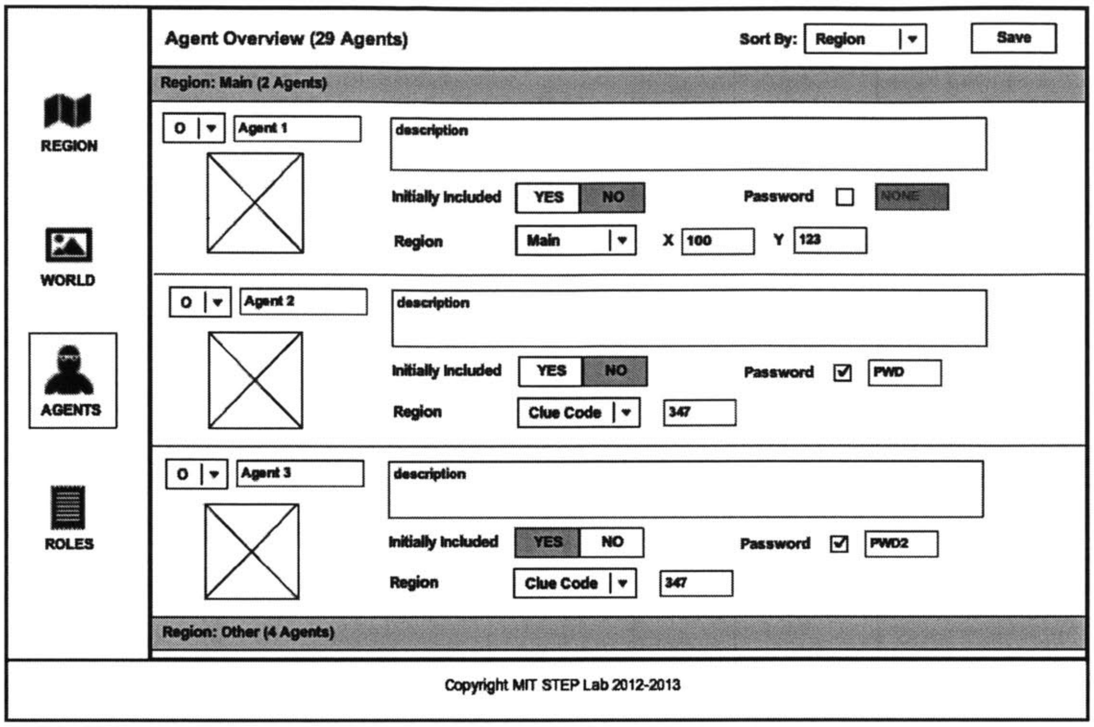
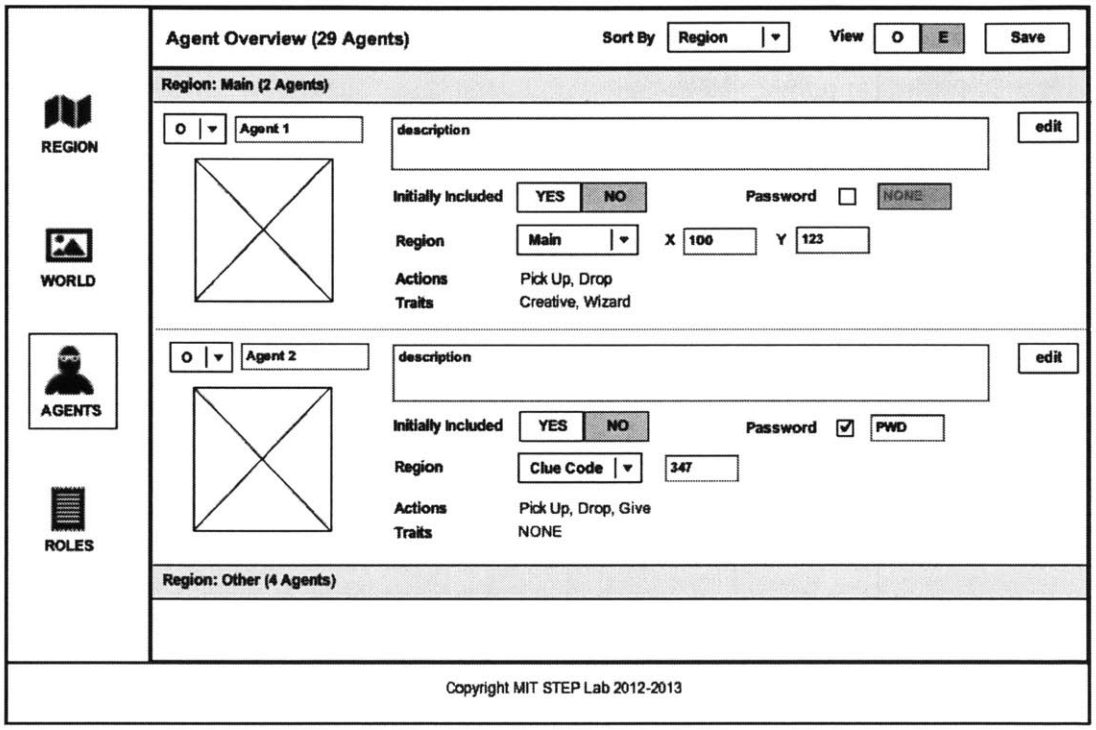


Figure 3-3: The initial wireframe mockup for the agent overview

location/clue code, and indication of whether the agent is initially included in the game, and whether it is password protected. These fields are designed to be editable rather than just viewable in order to aid in the goal of being able to efficiently update multiple agents in a minimum number of step. (See Figure 3-3)

After the feedback from the TaleBlazer team, the overview design was changed to also include some non-editable fields such as a list of traits and actions that can be edited by pressing an edit link that redirects the user to the scripts pane and settings editable view. The description was also moved to be below the agent image to follow the WYSIWYG layout of the agent page on the mobile device. These changes can be seen in the second round of mockups in Figures 3-4 and 3-5. In the last mockup seen in Figure 3-5, the material was condensed so that at least two tiles could appear in each row on the screen in an attempt to maximize the amount of information seen without making the screen too crowded.

The design from Figure 3-5 is the design that was used for the first computer



Sort By: Region

Region: Main (2 Agents)

O | Agent 1 | description

Initially Included: YES NO — Password ☐ NONE

Region: Main — X 100 — Y 123

Actions: Pick Up, Drop

Traits: Creative, Wizard

O | Agent 2 | description

Initially Included: YES NO — Password ☑ PWD

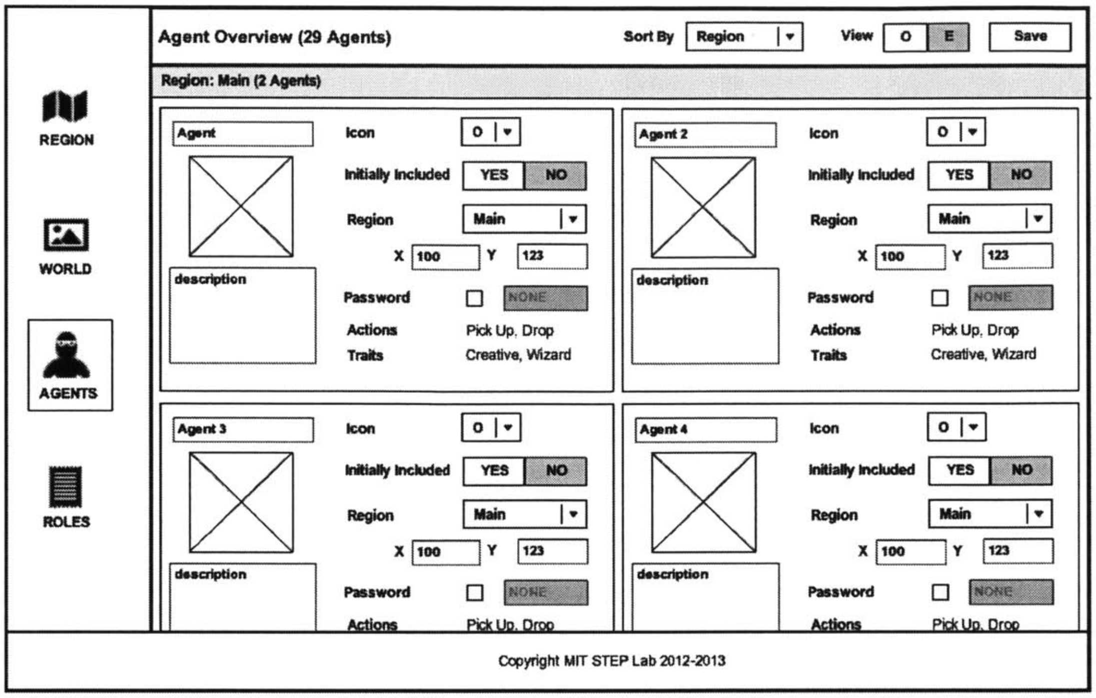
Region: Clue Code — 347

Actions: Pick Up, Drop, Give

Traits: NONE

Region: Other (4 Agents)

Figure 3-4: The second wireframe mockup for the agent overview with traits and actions included



Agent Overview (29 Agents) Sort By Region View O E

Region: Main (2 Agents)

Agent	Icon	O
description	Initially Included	YES NO
	Region	Main
	X	100
	Y	123
	Password	☐ NONE
	Actions	Pick Up, Drop
	Traits	Creative, Wizard

Agent 2	Icon	O
description	Initially Included	YES NO
	Region	Main
	X	100
	Y	123
	Password	☐ NONE
	Actions	Pick Up, Drop
	Traits	Creative, Wizard

Agent 3	Icon	O
description	Initially Included	YES NO
	Region	Main
	X	100
	Y	123
	Password	☐ NONE
	Actions	Pick Up, Drop

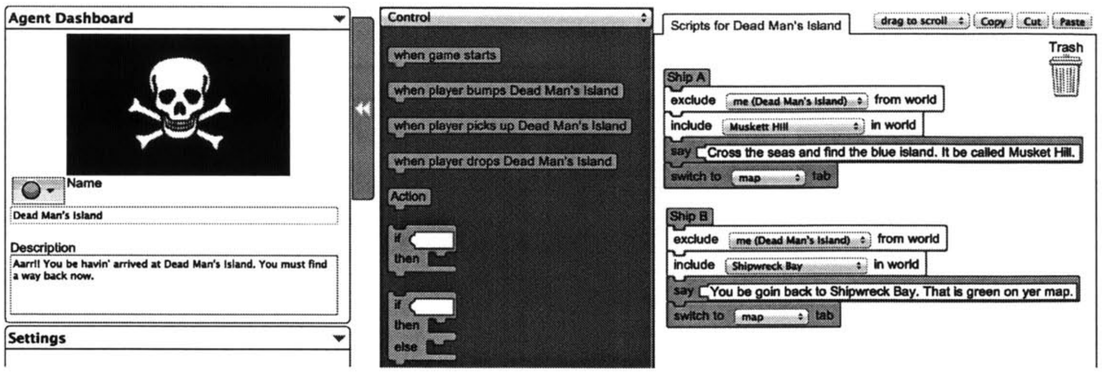
Agent 4	Icon	O
description	Initially Included	YES NO
	Region	Main
	X	100
	Y	123
	Password	☐ NONE
	Actions	Pick Up, Drop

Figure 3-5: The tiled wireframe mockup for the agent overview with traits and actions included

prototype of the agent overview. The revised editor now includes a version of the agent overview that is derived from this basic design. The final implementation differs slightly from the mockup in Figure 3-5 as a result of feedback from the testing performed as described in section 4.1.

3.4 Printable Game Summary

Similar to the agent overview, the revised Editor includes a game summary to give game designers a 'big picture' view of their game. This new game summary enables the user to easily share and print a hard copy of their games. There are no images included in the game summary in order to make the print view more succinct and text focused since most of the users of TaleBlazer expressed the desire to have a text proofreading tool rather than a more visual overview tool. Unlike the agent overview, this game summary is not editable and as such is a read-only view. After consideration of an editable overview, it was decided that having editable form fields


Agent Dashboard
Settings
Control
when game starts
when player bumps Dead Man's Island
when player picks up Dead Man's Island
when player drops Dead Man's Island
Action
if
then
else
Scripts for Dead Man's Island
drag to scroll
Copy
Cut
Paste
Trash

Name
Dead Man's Island

Description
Aarr!! You be havin' arrived at Dead Man's Island. You must find a way back now.

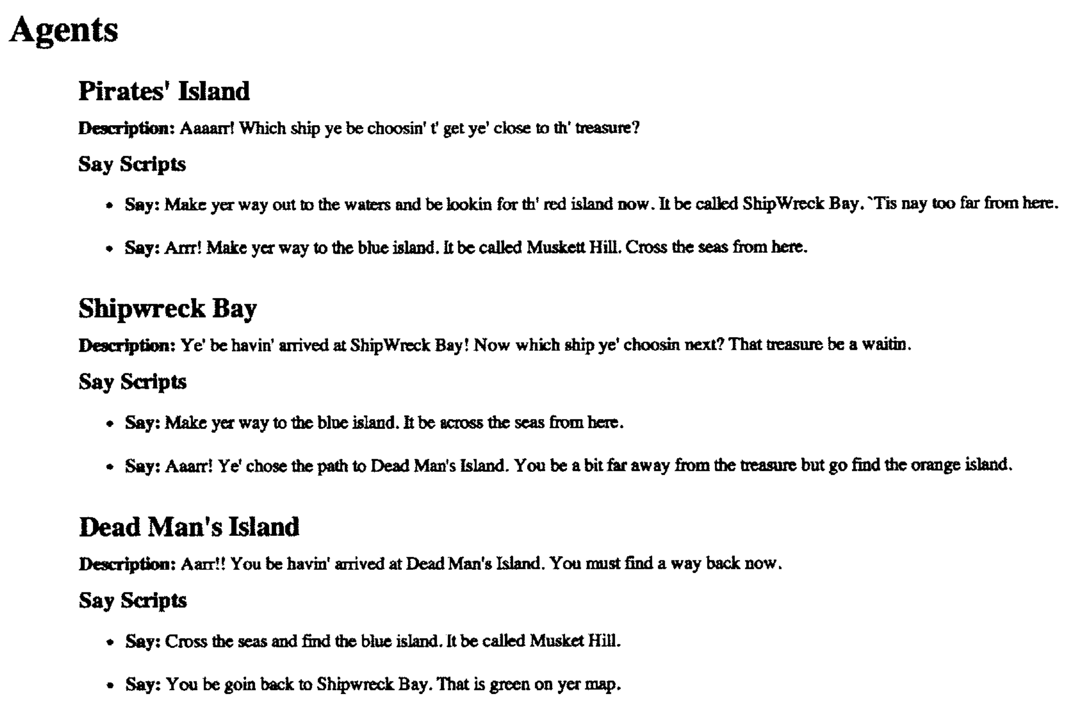
```
Ship A
exclude me (Dead Man's Island) from world
include Muskett Hill in world
say Cross the seas and find the blue island. It be called Musket Hill.
switch to map tab

Ship B
exclude me (Dead Man's Island) from world
include Shipwreck Bay in world
say You be goin back to Shipwreck Bay. That is green on yer map.
switch to map tab
```

Figure 3-6: Example editor view for the Dead Man's Island agent with name and description test on the left and scripts on the right. There are two say scripts that are later represented in the game summary view

in this instance would likely confuse game authors, having lost all of the organization that is provided by the tabbed structure of the current editor. Having one large editable form for the game would be even more difficult to navigate than the current structure. A read-only summary is space efficient, allowing the user to see more on their screen at once and it is easily printable and sharable with others. The game designer can easily use this overview to search through all the text in the game, although they need to return to the tabbed interface to change the text. Having a printable view makes proofreading of all text in the game simpler since all of the text is located in one place rather than across multiple tabs, script blocks, and rich text editor pop-ups as it was in the original editor.

The game summary includes details about the world, agents and roles. For every entity in the game (the world, all of the agents, and all of the roles), there is listed the name of the entity, the description, the names and content of all of the actions associated with the entity, the names of all the traits for the entity, and the contents of all the say blocks in the entity's scripts. You can see an example of how the descriptions and scripts for the Dead Man's Island agent in Figure 3-6 is recorded in the agents section of the game summary in Figure 3-7. The game summary aims to be a useful overview of all of the text in the game allowing the game designer to quickly see all of this text in one place to allow for a simple read over.



Agents

Pirates' Island

Description: Aaaarr! Which ship ye be choosin' t' get ye' close to th' treasure?

Say Scripts

- **Say:** Make yer way out to the waters and be lookin for th' red island now. It be called ShipWreck Bay. `Tis nay too far from here.
- **Say:** Arrr! Make yer way to the blue island. It be called Muskett Hill. Cross the seas from here.

Shipwreck Bay

Description: Ye' be havin' arrived at ShipWreck Bay! Now which ship ye' choosin next? That treasure be a waitin.

Say Scripts

- **Say:** Make yer way to the blue island. It be across the seas from here.
- **Say:** Aaarr! Ye' chose the path to Dead Man's Island. You be a bit far away from the treasure but go find the orange island.

Dead Man's Island

Description: Aarr!! You be havin' arrived at Dead Man's Island. You must find a way back now.

Say Scripts

- **Say:** Cross the seas and find the blue island. It be called Musket Hill.
- **Say:** You be goin back to Shipwreck Bay. That is green on yer map.

Figure 3-7: Example game summary view of the agents in the game. The output for the Dead Man's Island agent shows all of the text from Figure 3-6

3.5 Error Checker

To prevent users from having runtime errors as they play their games, modifications to the editor were made to include basic error checking to alert game designers to common mistakes. The types of common errors were generated by the TaleBlazer team from personal experience creating games and from previous comments from TaleBlazer users at our partner institutions. The user poll listed a couple of these common errors and the users voted on which ones were most pressing for them. The preliminary computer implementation of the error checker checks for the following errors and warnings:

- Blocks referring to deleted entities
- Mobile settings set to numeric password keyboard for games that include agents with non-numeric passwords
- Mobile settings set to numeric clue code keyboard for games that include agents with non-numeric clue codes
- Orphaned clue codes
- Using the default clue code
- Duplicate clue codes
- Orphaned regions
- Orphaned agents

And the subsequent errors and warnings were added after a round of iterative testing as described in Section 4.3:

- Missing arguments in a block
- Empty text actions
- Empty script and video actions

The user is notified if they have any of these errors anywhere in their game and clicking on the error will bring users to the site of the error. The user is not prevented from saving their game due to the presence of any of these errors. This allows users to be aware of the problems they face with their game but still allows them to postpone fixing their errors until future sessions or continue onwards conscious of the risks.

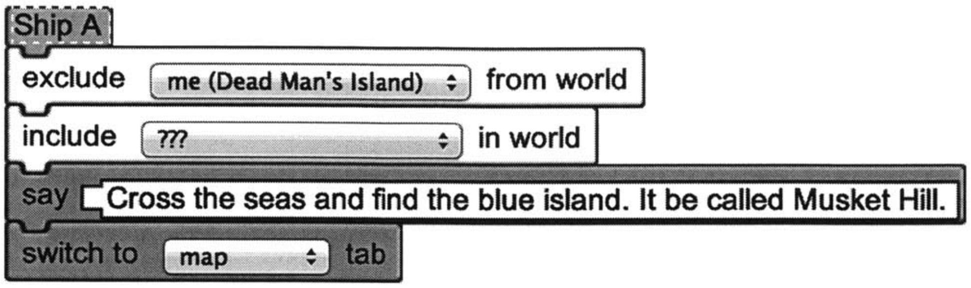
These error checks reflect some of the most common mistakes that absorb the most of the user's time when debugging. All of these errors originally required the user to look through all of their scripts for all of their agents, roles, and world, which is both time consuming and frustrating to attempt. These simple checks speed up the debugging process immensely and add to the efficiency of the game editor.

The error checker splits alerts indicating identified issues into 'warnings' and 'errors'. This partitioning is done to inform the game designer about the severity of the issue(s) with the game. Errors are defined as issues that will break the game when it is played on the mobile device. Warnings are issues that are most likely mistakes but won't necessarily break the mobile game during runtime. This segregation allows the game designer to filter the issues presented.

3.5.1 Errors

Blocks Referring to Deleted Entities

Currently, some of the TaleBlazer blocks contain drop down menus where an agent or role or other game entity can be selected. These blocks, such as the "Move <agent> to <region>" block or the "Include <agent> in world" block must have an agent or region selected in order to function. The common error with these blocks occurs when the selected agent or region or role gets deleted and the block no longer has a valid reference. When a block refers to a deleted entity, the name of the entity selected is converted to '???' as seen in the "Include <agent> in world" block in Figure 3-8. The error checker crawls through all of the scripts in the game and ensures that all of the selected entities are still valid entities in the game.



```
Ship A
exclude me (Dead Man's Island) from world
include ??? in world
say Cross the seas and find the blue island. It be called Musket Hill.
switch to map tab
```

Figure 3-8: Example of a block with a reference to a deleted entity that is replaced by '???'

Mobile Settings Set to Numeric Password Keyboard with Non-Numeric Passwords

TaleBlazer games have the ability to make agents that are password protected. On the mobile game, a keyboard pops up when a password needs to be typed in. The game has a mobile settings panel where the game designer can specify if this keyboard should be numeric or alphabetic by default. Some mobile devices function such that a numeric keyboard can never be switched back to an alphabetic keyboard. This error on the mobile device will prevent the player from completing the game. In order to prevent this occurrence, the error checker goes through all the passwords in the game if the keyboard is set to numeric and ensures that there are no agents with non-numeric passwords.

Mobile Settings Set to Numeric Clue Code Keyboard with Non-Numeric Clue Codes

TaleBlazer games have the ability to make agents with clue codes for locations instead of GPS locations. Similar to passwords, on the mobile game, a keyboard pops up when a clue code needs to be typed in. Similar to the password, the mobile settings panel has a setting for the default keyboard type for clue codes. When this setting is set to numeric and there are non-numeric clue codes the mobile game breaks in the same way it does for passwords as described above. The error checker also goes through all of the agents to ensure that there are no agents with non-numeric clue codes if the default keyboard is set to numeric.

Empty Script and Video Actions

When the game designer creates actions for an entity, they are allowed to choose between having a script action, a text action, or a video action. When the designer chooses to have a script action, they are prompted to choose a named script to associate with the action from a dropdown. When the designer chooses a video action, they are prompted to upload the video for the action. If either of these actions don't have the required script or video selected, the game will have nothing to play or execute when running on the mobile. The error checker provides an error for any script or video action that does not have the correct associated content.

3.5.2 Warnings

Orphaned Clue Code

Every agent has a location, which is either a coordinate in a region or is a clue code. The game player interacts with clue code agents by typing its code into the Clue Code tab on the game. In order to be able to type in a code, the game designer must include the Clue Code tab either in the mobile tabs settings for the world or during the course of the game play by using the "Include clue code tab" block. The error check ensures that the Clue Code tab gets included at some point in the game if there are agents in the game with clue codes. The error checker puts out a warning if there are agents with clue codes but the Clue Code tab never appears since this means these agents will never be accessible in the game.

Default Clue Code

Every agent by default has a location on the top left corner of the default region. When the game designer switches the location to use a clue code instead, the clue code is automatically set to a default clue code. It is not likely that the game designer would ever intend to have the clue code for an agent be the automatically generated default code, so the error checker provides a warning for any agents that use the default clue code.

Duplicate Clue Code

The game player interacts with clue code agents by entering its clue code into the Clue Code tab of the TaleBlazer game. Entering this code should call up the agent with that code currently included in the game. If there are two agents included with the same code, it is not clear which agent will be 'bumped'. Any duplicate clue codes are most likely a mistake that the game designer overlooked and the error checker looks through all the agent's clue codes and gives a warning for any duplicates. This is considered a warning since there are some instances where a game designer could intentionally have two agents with the same code, for instance, in a game where only one of the two agents is only ever included in the game at once and the game designer wants the code to bump whichever of the two agents is included at the time.

Orphaned Regions

For every TaleBlazer game, the game designer is allowed to specify multiple regions and place agents in any of these regions. One region is marked as the default region and the game player always starts in this region. In order to go to another region, the game designer must include a "move <player> to <region>" block. If there is a region in a game that is not the default and there is no "move <player> to <region>" for that region then the region can never be accessed by the game player. This is most likely not the intent of the game designer so the error checker ensures that there is a "move <player> to <region>" block for every non-default region and prints a warning if there is an unreachable "orphaned" region.

Orphaned Agents

All TaleBlazer agents have a property specifying whether they are to be included in the game at the start of the game. All of the agents that are included at the start of the game and are in non-orphaned regions are reachable by the game player at some point in the game. If an agent is not accessible from the start, the only way to interact with the agent is by including them somewhere during the game with a "include <agent> in world" block.

If the agent is not included at the start and there is no "include <agent> in world" block for it, then the agent will never be accessible during the game. This is most likely not the intent of the game designer, so the error checker provides a warning for every agent that is never included at some point during the game play.

Missing Arguments

Most of the blocks in TaleBlazer have arguments that need to be filled in. For example, the "arg1 < arg2" block should have values filled in for arg1 and arg2 so that they can be compared.

Currently, most of the blocks have default responses if there are missing arguments, but this default is hard coded into the block evaluator at run-time and is not evident to the user at design-time. Generally, if a block has a missing argument it is because the game designer mistakenly left it blank. This is classified as a warning since having missing arguments won't break the game, but it is most likely unintended by the game designer and could have non-obvious outcomes. The error checker crawls through all of the scripts in the game and ensures that each script has all of the arguments expected from the definition of the block specified.

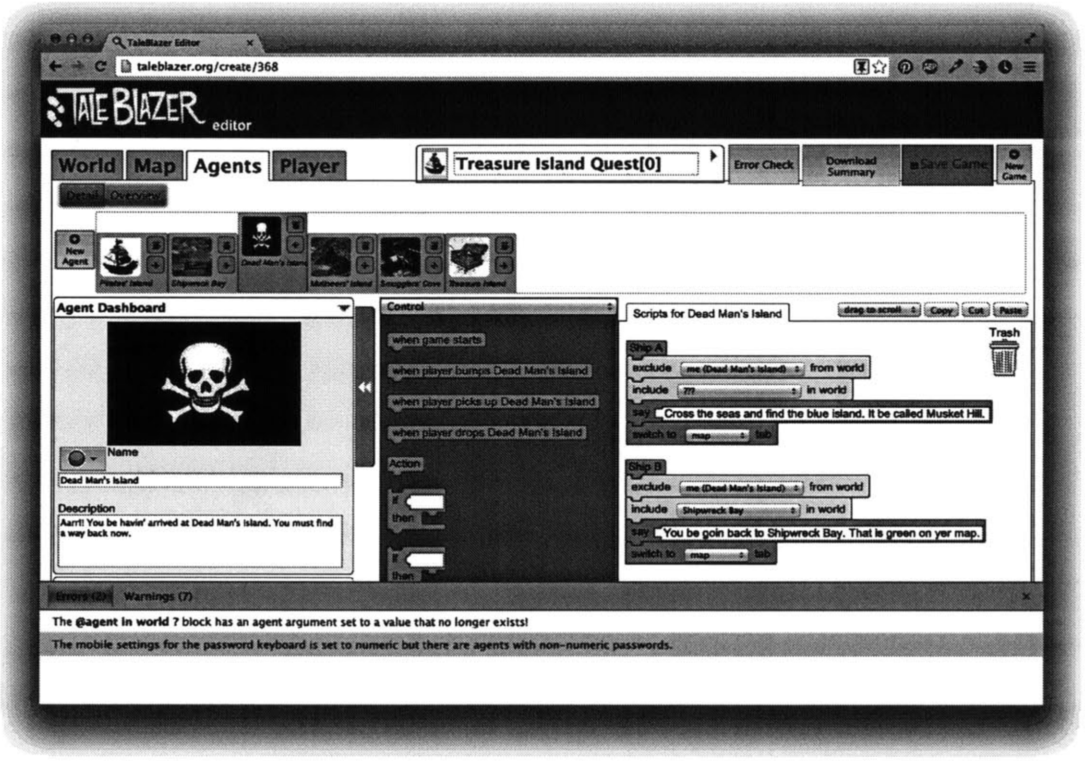
Empty Text Actions

When the game designer creates actions for an entity, they are allowed to choose between having a script action, a text action, or a video action. When they choose a text action, they are prompted to enter text that will appear when the action occurs. The error checker prints a warning for any text action that is left blank (without text) since that is most likely an oversight by the game designer, which results in an empty string showing up on the screen during the game. This, unlike the script and video actions, is considered a warning since it does not break the game, but is rather something that was just not desirable in most circumstances.

3.5.3 Presentation of Errors

This error checker is intended to be an additional feature that the game designer can take advantage of to debug their games, but does not force the designer to fix all problems before saving the game. The user can run the error check by pressing the Error Check button that appears in the top right of the editor. Pressing this button opens up the error checker window with the listed errors and warnings. The error checker is located in a place on the screen so that the user can see the list of errors while still seeing most of the editor panels so that they can fix the presented errors. To keep with external consistency for most error checkers and debuggers in programming IDEs, the error check in TaleBlazer is located in a fixed position box at the bottom of the screen. This is also consistent with the setup for the console debugger and development tools that are present in most web browsers. The user can close the error checker by pressing the 'X' in the top right corner. The first implementation of the error checker can be seen in Figure 3-9.

The errors and warnings are split into two different tabs in the bottom panel in order to provide a separation between these two types. The errors are presented on alternating color lines and can be clicked on to direct the user to the correct location to fix the error. Clicking on an error does not close the error-checking box so that the user can still reference the error list as they are correcting the problems in the editor. The other considered method for presenting errors was to have a popup presentation of all of the errors. This method was rejected due to the problems with viewing both the error and the editor at the same time. This also has the disadvantage of not being consistent with other debuggers such as Eclipse. Since TaleBlazer is an introduction for many young adults to software development, it is appropriate to present tools in a manner consistent with professional development environments in order to train them for any future work.



Name

Dead Man's Island

Description

Aarr!! You be havin' arrived at Dead Man's Island. You must find a way back now.

Ship A
exclude me (Dead Man's Island) from world
include ??? in world
say Cross the seas and find the blue island. It be called Musket Hill.
switch to map tab

Ship B
exclude me (Dead Man's Island) from world
include Shipwreck Bay in world
say You be goin back to Shipwreck Bay. That is green on yer map.
switch to map tab

The **@agent in world ?** block has an agent argument set to a value that no longer exists!

The mobile settings for the password keyboard is set to numeric but there are agents with non-numeric passwords.

Figure 3-9: View of the first iteration of the error checker. The error checker popup appears across the bottom of the screen and the Error Check button appears on the top left

3.6 Potential Flow and Interface Redesign

The original editor interface optimizes for efficiency for the super user, but this interface has the downside of appearing overly complicated to a new user. The usability of the current feature set implemented in the editor could be improved to increase the learnability of the system for new users while still maintaining the level of efficiency or even improving the efficiency for the super user.

The new TaleBlazer interface could be designed in many ways, some more drastic of a change than others. Many designs were proposed over the course of this update, but most were not implemented due to lack of consensus amongst the TaleBlazer team as to which changes would be too dramatic for the current users of TaleBlazer. The design that was settled upon deviates very little from the original editor design but does succeed in improving the main view. This design was chosen since it addresses the problem mentioned by users at Global Kids of having a busy layout and non-modern feel that is incongruous with the other web applications users are now accustomed to using. It also avoids confusing the set of current users who are accustomed to the current workflow of TaleBlazer but still helps new users feel a little more comfortable with the application. The layout of the editor user interface was rearranged and restyled to improve the simplicity. Originally the user was presented with cluttered views and oftentimes distracting colors. The new streamlined appearance of the interface additionally aids in comprehension of which features are most important. This is done by reorganizing the arrangement of the property form fields to emphasize the most important properties and by putting more advanced settings in less prominent positions. There are many other changes that were discussed that would potentially increase the usability of the editor, but after discussion with the TaleBlazer team, the more radical changes to the interface were postponed in favor of adding the new features described above. It was decided after careful consideration that any major overhaul of the flow of the editor would need lots of time and careful attention so as to not completely confuse our already large set of regular users and should be saved for a future iteration of editor updates.

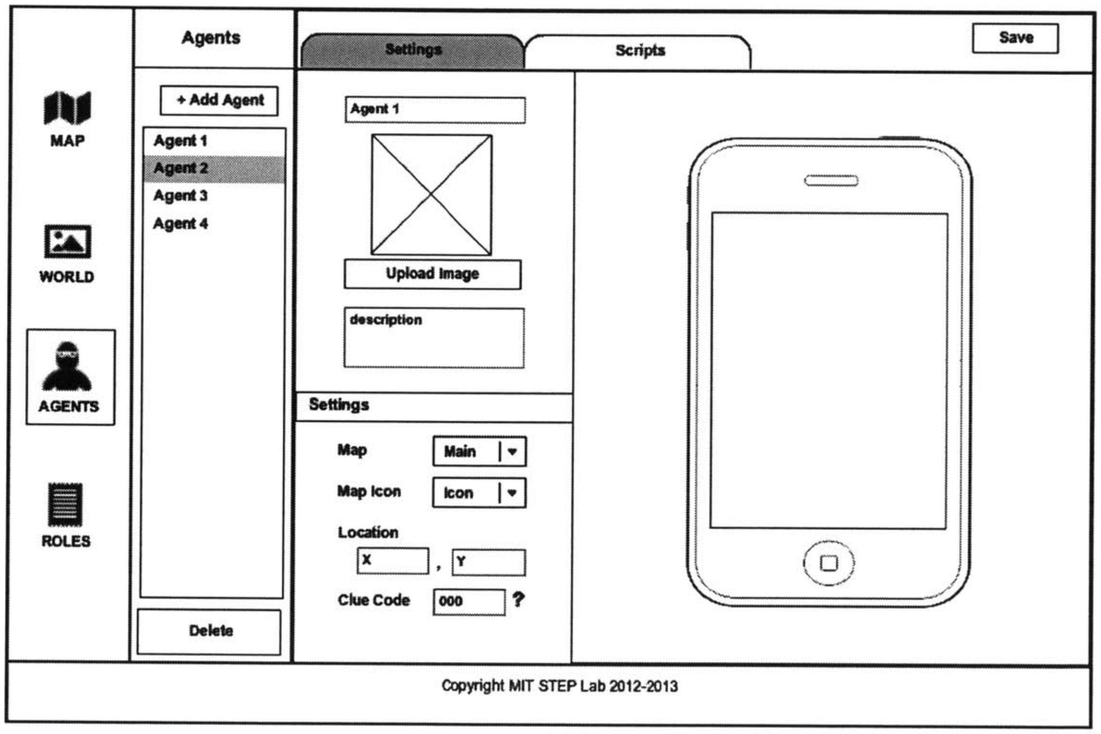


Figure 3-10: Possible wireframe mockup of the TaleBlazer agent editor page with an embedded mobile emulator

Ideally, in a future design, a new user would be able to play around with the different controls in the editor to get a sense for what each one does, but in the TaleBlazer editor there is a lack of direct feedback since most of the changes aren't obvious until you run the compiled game on a mobile device. One could imagine TaleBlazer having an emulator embedded in the editor in the future, as seen in Figure 3-10. In the absence of a mobile game emulator for feedback, the interface of the editor needs to have a very clear indication of what every control does to increase the learnability of the system. This could be done in the future with inline help dialogues and an attempt to use WYSIWYG layouts where possible, while still maintaining external consistency with other standard editing software. To increase internal consistency, the vocabulary being used in the editor should be kept to a small subset so that the user isn't overwhelmed by multiple terms being used for similar concepts, such as map vs. region or player vs. role. One possible design, as seen in Figures 3-11 and 3-12, is to move the horizontally scrolling ribbon of agents and regions and roles to be a vertically scrolling list on the left hand side of the screen. This would be externally consistent with most lists of objects on other websites and would allow users to select multiple entities at once in order to clone or delete groups of entities rather than using the current method of pressing the delete or clone button that is present in each tile for each entity in the ribbon. This would aid in the efficiency of deleting or cloning multiple entities. This would also aid in the simplicity of the view by removing excess buttons, which allows the user to focus on the more critical buttons in the view.

Another possible design is to divide the scripts pane and the settings pane into two separate tabs, as seen in Figure 3-13. In the original editor the scripts pane is very constricted width-wise since there are too many panels splitting up the width of the screen. By creating a larger area for the scripts, the user will have a more complete view of their scripts, which will hopefully allow them to have better organization and comprehension of their game logic. Better organization of the scripts will hopefully lead to fewer mistakes or oversights. It will also make the experience of editing and creating a game more pleasant by mitigating the need for frequent horizontal and vertical scrolling.

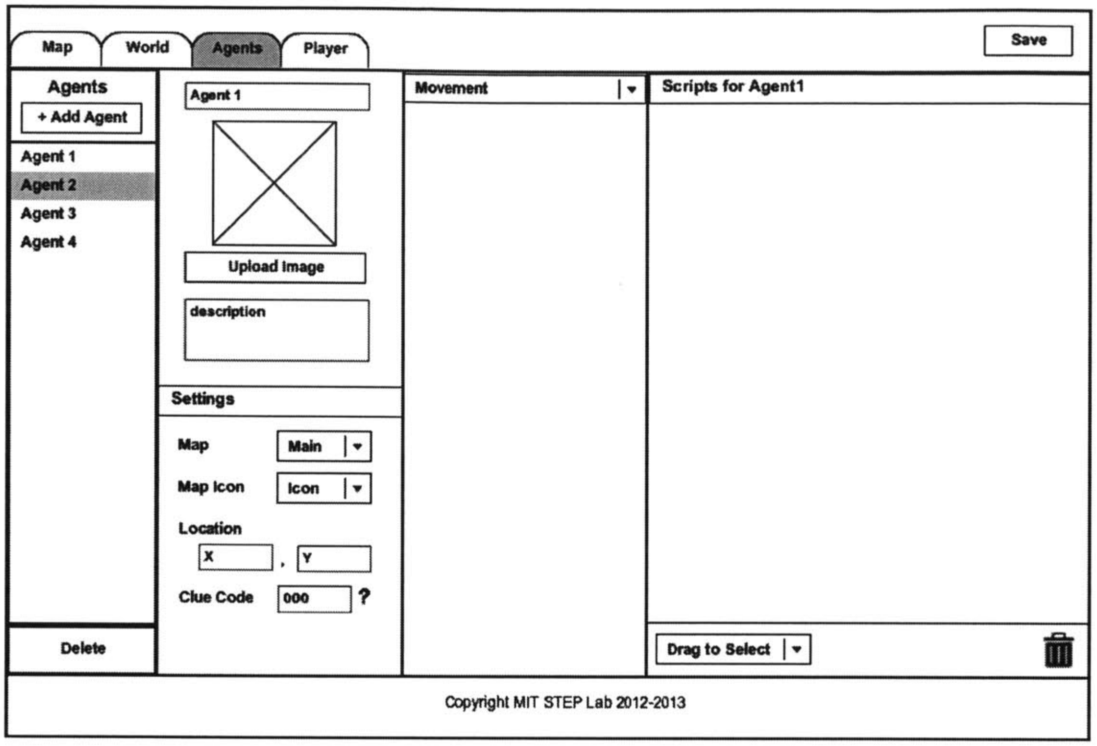
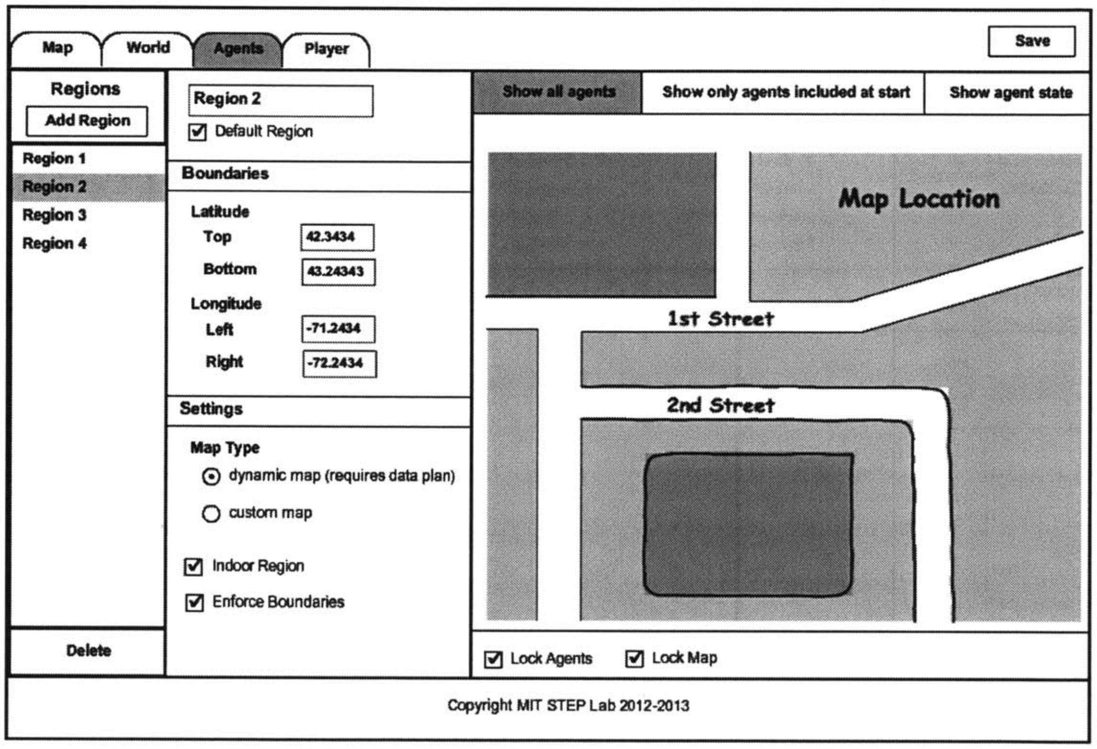


Figure 3-11: Initial wireframe mockup of the editor agent page redesign

For this layout seen in Figure 3-13 with the separate tabs for settings and scripts, the tab bar for switching between the Map, World, Agents, and Roles tabs is moved down to the left side. This move is to help prevent the possible confusion of having two layers of horizontally listed tabbed panels. This also means icons could be added to the selection bar, which would be nice for younger children who are still accustomed to using very image based websites that were designed to accommodate people with limited reading skills. While it is assumed that everyone using TaleBlazer has a good grasp of reading, it would still make the website more externally consistent with other sites that are not making that assumption but are intended for children of the same age.



Region 2

✓ Default Region

Boundaries

Latitude

Top	42.3434
Bottom	43.24343

Longitude

Left	-71.2434
Right	-72.2434

Settings

Map Type

[✓ dynamic map (requires data plan)] custom map

✓ Indoor Region

✓ Enforce Boundaries

✓ Lock Agents ✓ Lock Map

Figure 3-12: Initial wireframe mockup of the editor map page redesign

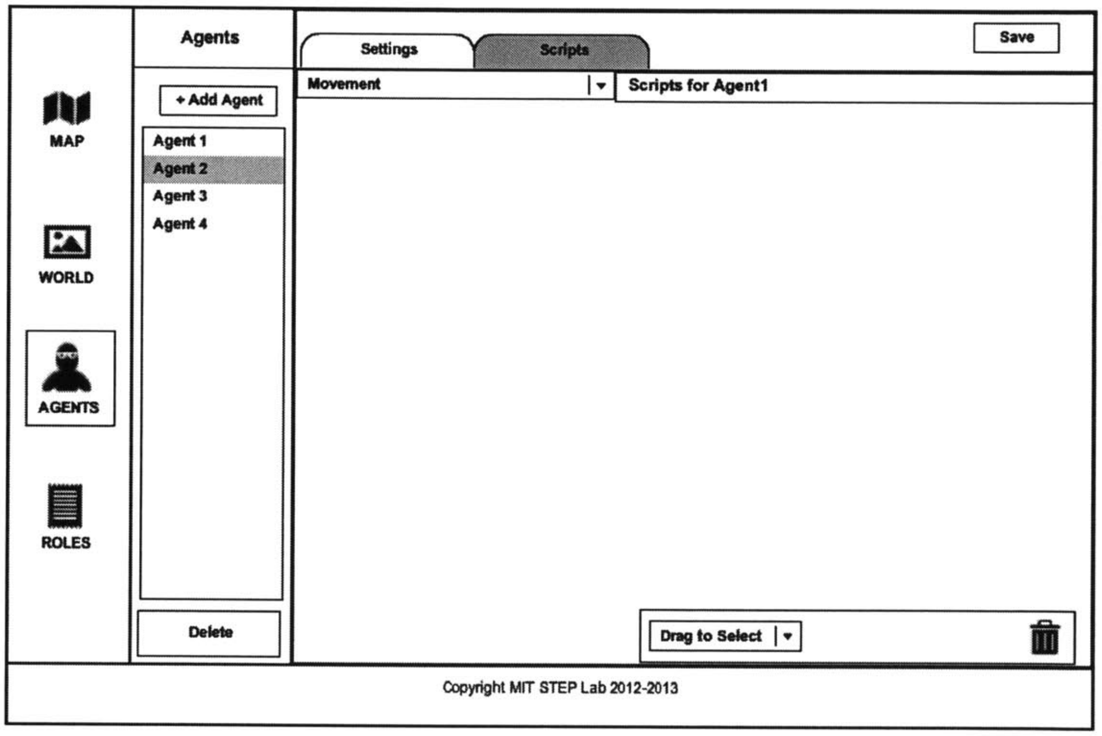


Figure 3-13: Alternate wireframe mockup of the editor agent page redesign featuring the new Map, World, Agents, Roles selection bar down the side and a tabulated view of the Settings and the Scripts

Chapter 4

Testing

Iterative design is the current best-practice process for developing user interfaces. It's a specialization of the spiral model for software engineering described by Boehm in "A Spiral Model of Software Development and Enhancement"[5]. The spiral model as applied to user interface design consists of cycling between design, implementation, and evaluation but with low cost at the early stages of the cycle and with higher fidelity and cost as the cycle spirals outwards. Since user interface design is focused on creating the most usable interface possible, it requires feedback and evaluation from the user to improve the system.

For all of the proposed improvements to the TaleBlazer editor there has been an iterative user-centered design and testing strategy. The original proposal for the new features came from the users of the system through watching students at the Global Kids afterschool program and through the requested features poll. After this first contact with the users, the original design for each selected new improvement was plotted out with wireframes and a task analysis was created through discussions with the TaleBlazer team leaders. The wireframes were then shown to other members of the STEP lab to get more user feedback. After the low fidelity, low cost stage of mocking up designs and proposing user interactions and supported features, there was a round of basic implementation of the features. The basic implementation resulted in a functional computer prototype that was then demoed and tested by the members of the STEP lab, all of which are users with limited familiarity with the

TaleBlazer editor. After receiving feedback from these users, the basic implementation was improved to fix any usability issues made apparent from preliminary testing. The second round of computer implementation was then tested via WebEx with one of TaleBlazer's regular users at the Missouri Botanical Garden.

All of the rounds of computer prototype user testing were formative evaluations where the users were observed interacting with the interface and were requested to think aloud. In each instance of testing, the users were told to try and check for and correct errors in a sample game as well as asked to review their game summary and sort and modify agents through the agent overview. After each round of testing, the qualitative usability issues were recorded and fixed before the next round of testing so as to get new feedback on ever iteration.

4.1 Agent Overview Testing

The preliminary design for the agent overview was created with multiple drawn mockups of the view as seen in Figures 3-3, 3-4 and 3-5. The main concern for the view was how many tiles should go across the screen and how much information should be included. After the preliminary mockups, the TaleBlazer team leaders agreed that having multiple agent tiles in each row would increase the number of entities on the screen, which would help with the goal of seeing more at once as an overview but without leaving it too cluttered. Additionally, it was decided that the traits and actions for an agent should be listed in the overview but should not be editable since the editing structure for the actions and traits would overly complicate the view. Instead, an edit button was added to the top corner of each tile, which brings the user back to the full view for the selected agent so that they can edit whatever they need to change there.

The second round of testing led to a change in the content provided on the agent tile. Two of the testers agreed that a link to the bump settings popup for an agent would be helpful since they oftentimes find themselves needing to update those settings. Another user also mentioned that it might be nice to summarize the content of the bump settings in the overview for each agent. For the next round, the agent tile incorporated a bump settings button on the agent tile that opens a popup for viewing and modifying the bump settings of the agent in question. One tester mentioned that the agent name field was not long enough for some of their longer agent names, so the styling of the agent tile was shifted slightly for the next implementation such that the tile stayed the same width but the width of the agent name input increased. The rest of the feedback was positive and many of the testers said that they could see themselves using the feature and finding it very helpful.

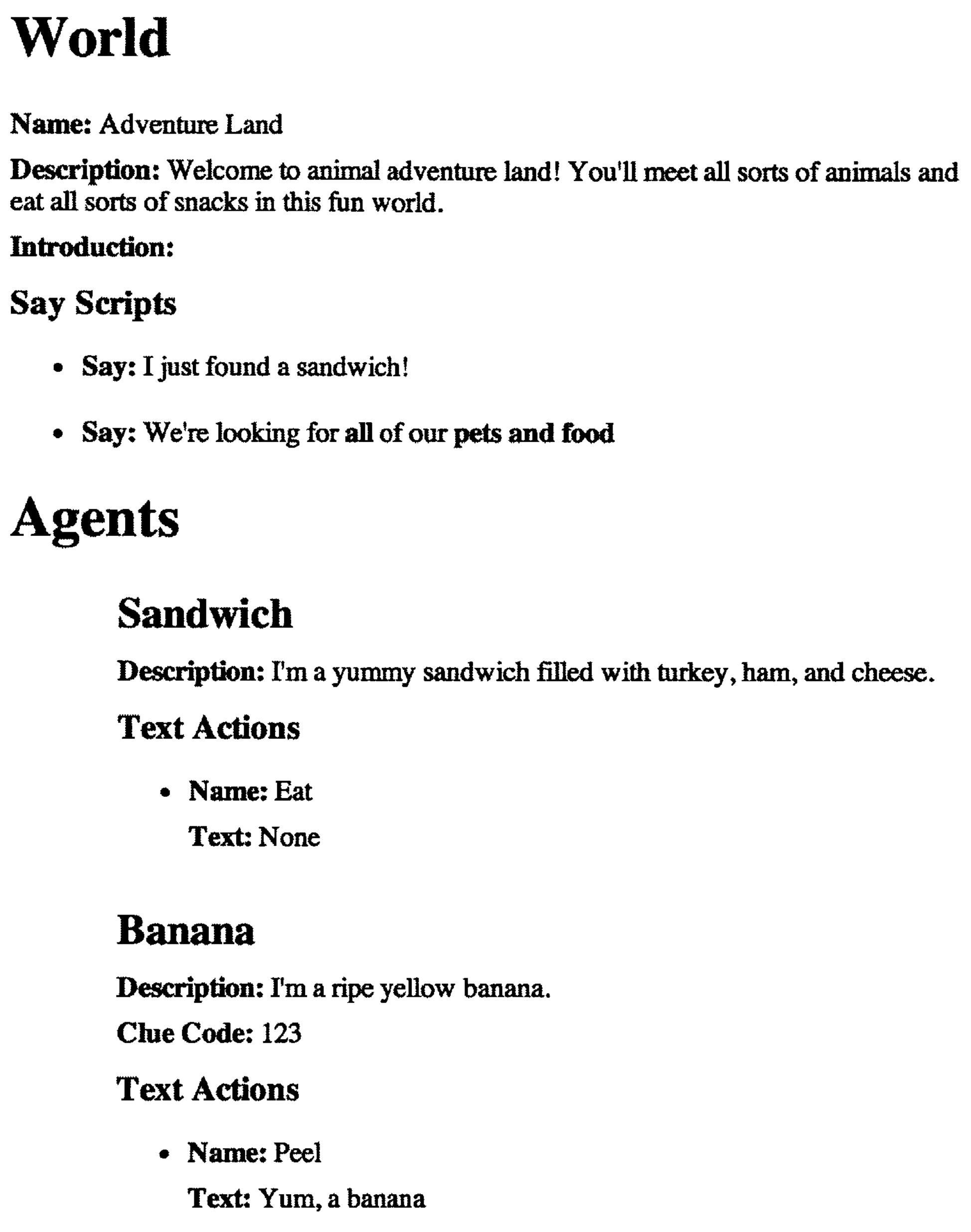
4.2 Game Summary Testing

The first design for the game summary generated a PDF file of the summary when the user clicked the game summary button. After the initial discussions of this method, the game summary button was changed so that it now saves an html version of the game summary. The user can now use the browser's search function on the summary file and the browser's print function. This change was due to the fact that some PDF viewers do not have a good capability for searching through text and one of the primary tasks that needs to be completed with the game summary is searching for the instance of an agent name or searching for any other specific game text.

During the second round of testing, one user expressed a desire to have less repetition of the text in the labels. Based on this feedback, all self explanatory and repetitive labels were removed in the next iteration. The first iteration can be seen in Figure 4-1 and the revised second iteration summary can be seen in Figure 4-2.

4.3 Error Checker Testing

The preliminary design for the error checker was based off of the browser console web development inspector and the Eclipse debugger. After the preliminary discussion, one question was whether or not to split errors into errors and warnings or to have just one tab. No consensus was reached in that discussion, so the two tab format of



World

Name: Adventure Land

Description: Welcome to animal adventure land! You'll meet all sorts of animals and eat all sorts of snacks in this fun world.

Introduction:

Say Scripts

- **Say:** I just found a sandwich!
- **Say:** We're looking for **all** of our **pets and food**

Agents

Sandwich

Description: I'm a yummy sandwich filled with turkey, ham, and cheese.

Text Actions

- **Name:** Eat
  Text: None

Banana

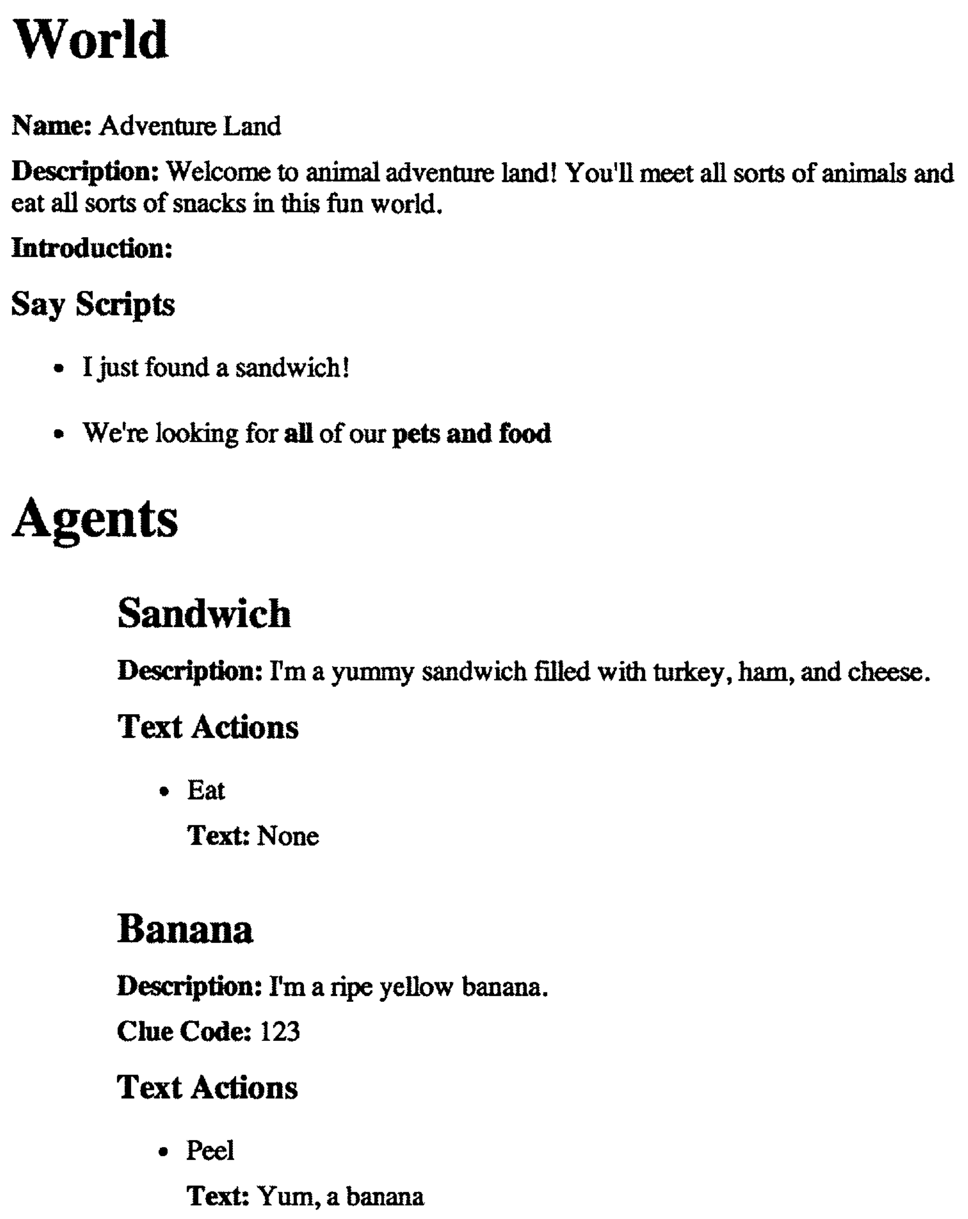
Description: I'm a ripe yellow banana.

Clue Code: 123

Text Actions

- **Name:** Peel
  Text: Yum, a banana

Figure 4-1: View of the first iteration of the game summary



World

Name: Adventure Land

Description: Welcome to animal adventure land! You'll meet all sorts of animals and eat all sorts of snacks in this fun world.

Introduction:

Say Scripts

- I just found a sandwich!
- We're looking for **all** of our **pets and food**

Agents

Sandwich

Description: I'm a yummy sandwich filled with turkey, ham, and cheese.

Text Actions

- Eat

  Text: None

Banana

Description: I'm a ripe yellow banana.

Clue Code: 123

Text Actions

- Peel

  Text: Yum, a banana

Figure 4-2: View of the second iteration of the game summary. All repetitive labels have been removed in this revised iteration

errors and warnings was kept for the next iteration of testing.

In the first round of testing, the users found it unclear that the errors listed in the checker were clickable. The shading on the selected tab for errors or warnings was also unclear to some of the testers. One tester also suggested adding in a refresh button and a "last updated" field on the error checker to let the user know that the error checker does not dynamically update as you change the game in the editor. As of the next iteration, the errors in the checker were styled with bold on the names of the incorrect entities in question and a hover effect was added to the line. Both of these styling changes helped the next round of users realize that the items in the list were clickable. A small refresh icon was also added with a statement of the last update time. The buttons for the selected tabs were changed to more standard button shapes, which also helped testers in the next round better understand which tab they were on. The final view after all of the testing iterations of the error checker can be seen in Figure 4-3.

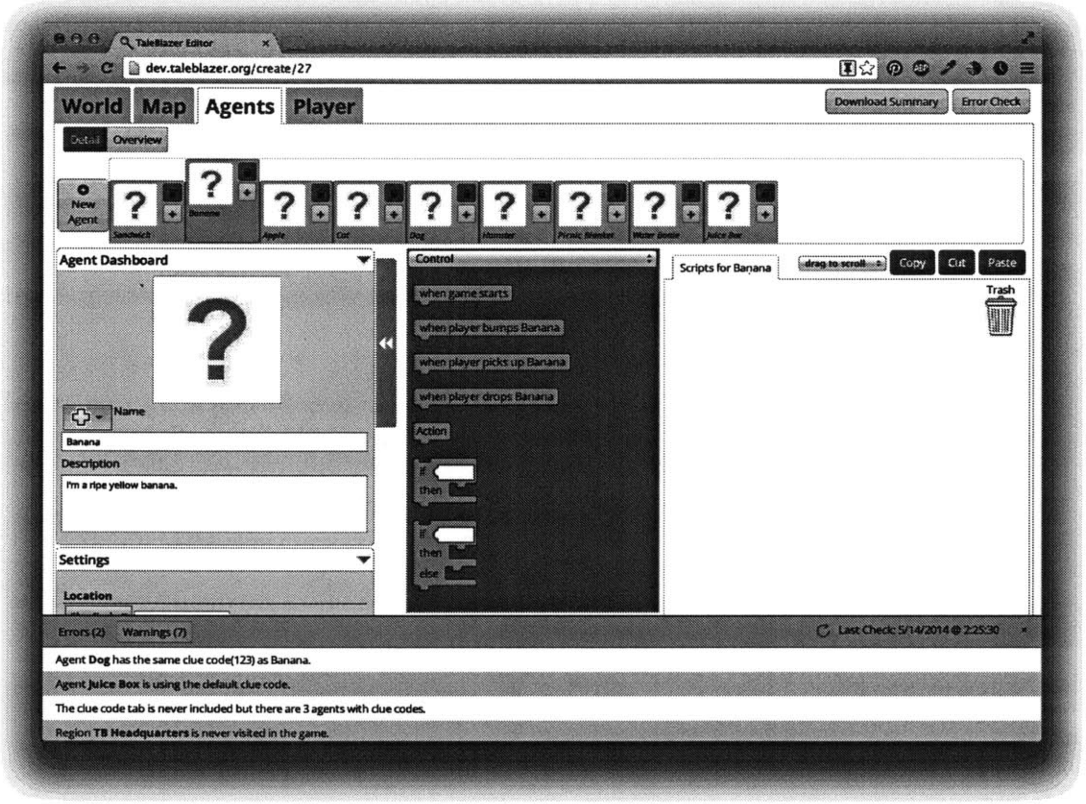
Finally, the users in the last test noted that it would be nice to have errors and warnings for three additional categories. These two new warnings and one new error were implemented for the next iteration:

- Warning for missing arguments in a block
- Warning for empty text actions
- Error for empty script and video actions

A detailed description of these errors can be found in Section 3.5

4.4 User Interface Redesign Testing

Over the course of discussions with the TaleBlazer team leads, multiple designs for the user interface redesign were presented. These include the mockups shown in Figures 3-11, 3-12, and 3-13 where the top tab bar has been moved to the side in many instances. Most of the designs also feature rearrangements of the script panels and the entity



Agent **Dog** has the same clue code(123) as Banana.

Agent **Juice Box** is using the default clue code.

The clue code tab is never included but there are 3 agents with clue codes.

Region **TB Headquarters** is never visited in the game.

Figure 4-3: The final view of the error checker after revisions from testing

ribbons. There was no consensus amongst the testers about the large changes planned for the large-scale redesign. The final improvements include a change in color palette, font, spacing between elements, label names, and form element alignment. These changes were decided upon due to a request from the users of having an application that appeared more modern and had a cleaner appearance. All of the changes in color were to minimize the number of accent colors and keep unimportant parts of the editor in neutral tones while having important buttons and highlighted selections in a contrasting accent pop color. This use of color directs the user's attention to the regions that are most critical for use in the editor. The additional padding, alignment, and spacing was added in response to the users feeling like there was too much information being crowded into one space. The new alignment and spacing provides each piece of the editor a little more separation from all of the other pieces, which helps the user distinguish between sections and feel less overwhelmed by the wealth of content being shown to them.

Chapter 5

Final Implementation

The TaleBlazer editor is composed of a server backend written in CakePHP and a frontend using html, css, and JavaScript. The editor also makes use of the Google Closure tool package and the JQuery library which both allow for a richer use of Javasdcript. The TaleBlazer editor also uses scriptsBlocks, a JavaScript library developed at MIT that is used across multiple projects in the STEP lab to create the TaleBlazer script editor. Most of the changes to the editor were written in html, css, and JavaScript and have very little interaction with the backend.

The frontend for the editor is set up using a set of views and underlying models. The editor also follows the observer pattern. There is a general event manager that has functions for triggering an alert and for subscribing to get events about the changes triggered from changes to the view, and subsequently, changes to the model. Every time a field is changed in the view, an event is triggered and the manager sends out an update to all the subscribers, which are generally the other views that use that same data that has been changed. For example, when the user changes the name of an agent, a rename event is sent out to every view that displays that agent name.

5.1 Agent Overview Final Implementation

The agent overview is a made up of two new views for the editor, one for the page of agent tiles with the sorting and one for the agent tile object.

The agent overview main page consists of a set of agent tile objects and a dropdown that allows the user to sort these tiles by name, date created, icon color, icon shape, region, and their custom sort order. There is a function that reorders the agent tiles in the view based on the selected sort order. The overview main page subscribes to change listeners for the element of the agent being sorted upon so that the sort order can be updated whenever that element of any agent gets changed. For example, if the sort order is being determined by icon color, the overview listens for agent icon change events and resorts the agents when it receives such an event. The agent overview main page also subscribes to the add and delete events for agents so that it can create or remove agent tiles for the new or deleted agent respectively.

The agent tile partial view is created for every agent model and the view contains fields for all of the pertinent elements of the agent and described in the design section. These include the name, description, image, traits, actions, location, password, and whether or not the agent is included at the start of the game. Each of these fields subscribe to rename and change events so that these fields are always consistent with the full agent view that can be reached by clicking the edit button on the top right corner of the agent tile. A change to any of these fields will also trigger the rename or change event for the agent being modified so that the other views using the information contained in that element get updated appropriately.

5.2 Game Summary Final Implementation

The game summary is a new view where most of the text content from the game is displayed in a simple read-only format. The summary gathers all the entities in the game, which consist of the world, all the agents, and all the roles. For every entity, the summary displays the name, the description, the actions for the entity, the names of the traits, and the content of any "say" or "say rich text" script blocks. Additionally, for every agent it includes passwords or clue codes when applicable.

There is a function that crawls through all of an entity's scripts in order to find any "say" blocks and grab their content. Each script block has a name in its definition

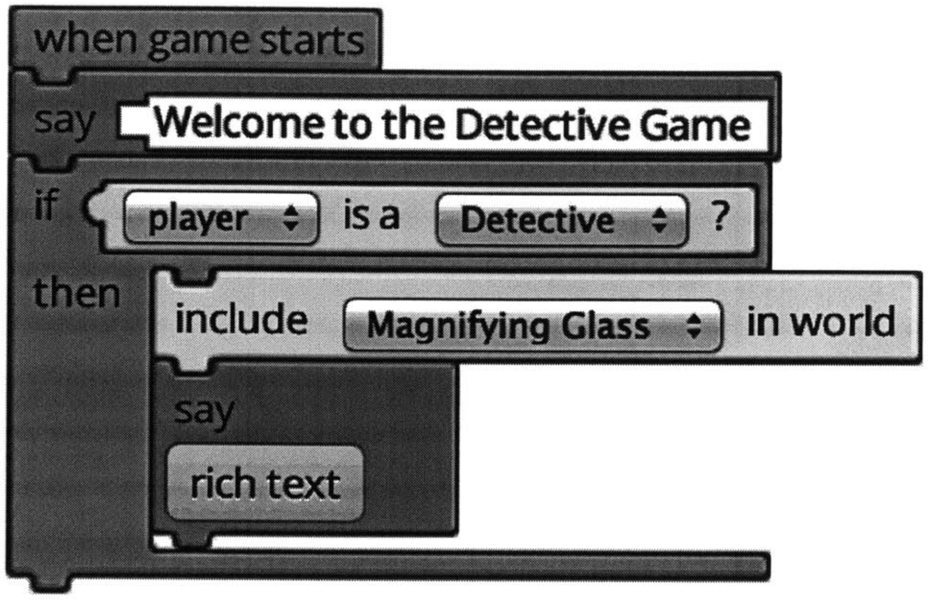


Figure 5-1: An example of a script with say blocks nested inside of other blocks

as well as a reference to any blocks it contains as well as the next attached block. The function starts with the root script blocks for each agent and recursively searches for say blocks by looking into any blocks contained within that block and then moves on to the next connected block, similar to a depth first search.

For example, with the script in Figure 5-1, the function would start at the root, the "when game starts block". This block is not a "say" block and does not contain any blocks nested in it, so the crawler would proceed onto the next block. This next block is a "say" block, so the text in this block would be pulled out and printed into the game summary. The next block is an "if/then" block, which does have blocks nested inside of it. The crawler would proceed to check inside the block in the "if" statement and subsequently in the blocks in the "then" statement. It would find no blocks of interest in the "if" statement and would move to the "include in world" block in the "then" statement and then onwards to the "say rich text" block, whose contents would be recorded in the game summary. At this point, the crawler would terminate since there are no next blocks.

The resultant text from all of the game entities is dynamically compiled into an

HTML layout when the game summary button is pressed. Pressing the summary button initiates a download of the HTML file, which can then be opened in the browser.

5.3 Error Checker Final Implementation

The main implementation of the error checker is a script crawler similar to the one used for the finding the "say" blocks in the game summary. This crawl script function also goes through all of the blocks in the game like a depth first search, but instead of searching for just "say" and "say rich text" blocks, the crawler takes in as an argument the function that specifies which error checks to execute for each block.

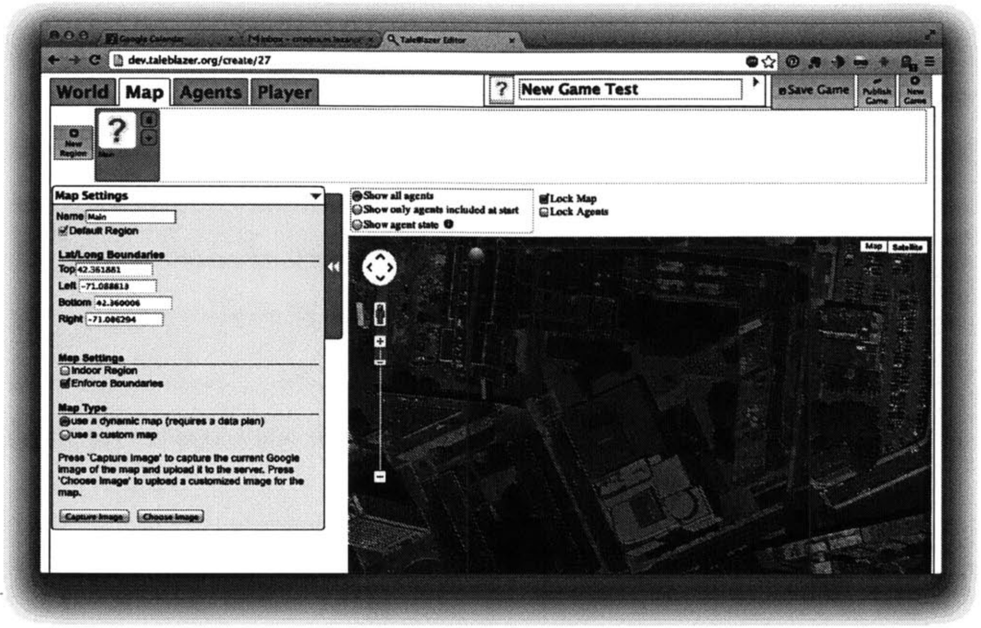
The error checker has functions to create links to the warnings and errors being created. Each one of these links is added to the error checker view that appears at the bottom of the editor. Each error is a link that on click changes the main editor view to be the page where the error is located. For example, if there is an error in the world settings, the error link will bring the user to the World tab and will scroll to the place on the page where the error is located.

Currently, the check function taken in as an argument to the crawler function looks for multiple blocks in order to find all of the errors and warnings specified in the design. One block the error checker looks for is the "include clue code tab" block. If this block isn't found anywhere in one of the scripts then a warning is produced. The check function also looks for the "set player region" block in order to update a list of all the regions that are visited throughout the game, which can then be compared to the list of regions in the game. A warning will be created for any regions that are not visited by the player at some point during the game. Similarly, the checker looks for any "Include <agent> in world" blocks and adds the agent to a list of agents that are made visible at some point in the game. If there are any initially hidden agents that are never included then the error checker creates a warning. The check function checks for missing arguments by looking for the dynamic arguments specified in the block dictionary and making sure the selected dynamic argument still exists in the game by cross checking the stored reference. Similarly, the checker uses the block specification in the dictionary to make sure that the block has all the arguments filled in. If any arguments are missing, the checker produces a warning link.

The error checker checks the actions of each entity in addition to its scripts. It checks the actions by checking the type of action and making sure there is the proper data provided for each action type. For example, if the action is a video action with no uploaded video associated with it, then an error link is added the error panel.

5.4 Interface Restyling Final Implementation

Since the design for the interface was never fully settled on, the implementation of the interface restyling was mainly done using some basic CSS styling changes. The colors of the interface were modified to provide a more visually appealing experience that is more consistent with modern web design practices. The field of the form and the text was re-spaced in order to provide a more simple and easily read layout. Generally the prior layout was very tightly packed and not well aligned, so these small usability concerns were easily remedied with CSS. A before and after view of the interface restyling of the map tab can be seen in Figures 5-2 and 5-3 respectively.



Map Settings

Name Main

☑ Default Region

Lat/Long Boundaries

Top 42.361881

Left -71.088813

Bottom 42.360006

Right -71.086294

Map Settings

☐ Indoor Region

☑ Enforce Boundaries

Map Type

● use a dynamic map (requires a data plan)

○ use a custom map

Press 'Capture Image' to capture the current Google image of the map and upload it to the server. Press 'Choose Image' to upload a customized image for the map.

● Show all agents

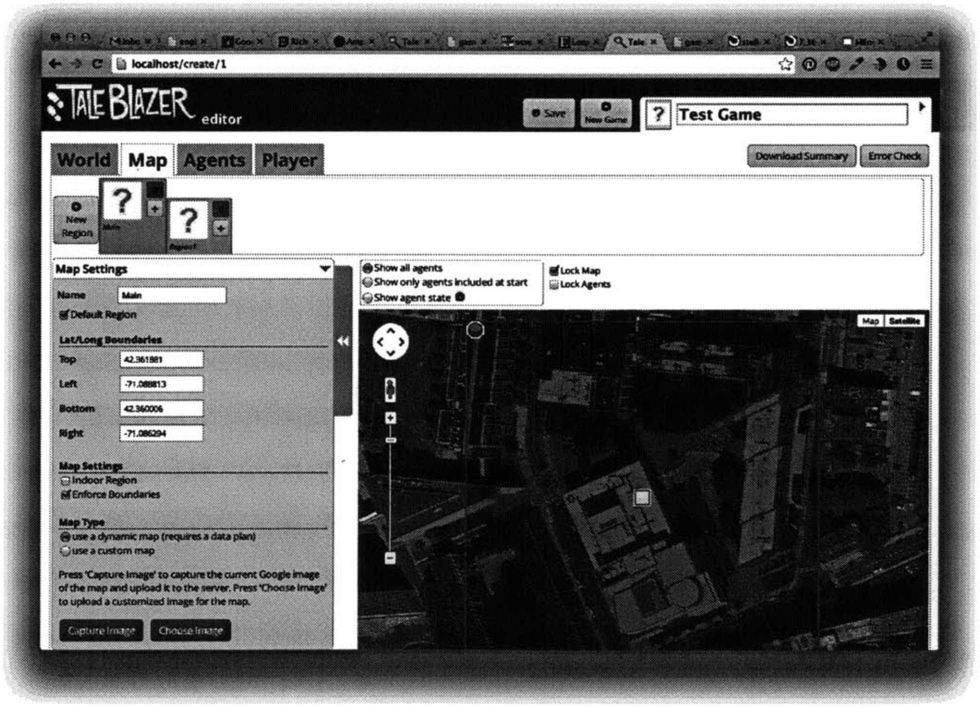
○ Show only agents included at start

○ Show agent state

☑ Lock Map

☐ Lock Agents

Figure 5-2: The view of the editor Map tab before this project's changes



Map Settings

Name Main

☑ Default Region

Lat/Long Boundaries

Top 42.361881

Left -71.088813

Bottom 42.360006

Right -71.086294

Map Settings

☐ Indoor Region

☑ Enforce Boundaries

Map Type

● use a dynamic map (requires a data plan)

○ use a custom map

Press 'Capture Image' to capture the current Google image of the map and upload it to the server. Press 'Choose Image' to upload a customized image for the map.

● Show all agents

○ Show only agents included at start

○ Show agent state

☑ Lock Map

☐ Lock Agents

Figure 5-3: The view of the editor Map tab after the final restyling changes

Chapter 6

Future Work

While the improvements to the editor have been significant, there is still more work to be done to make the tool even more user friendly. Some of the following changes would most likely make a marked improvement on the experience of a TaleBlazer user.

6.1 Editor Flow Redesign

One important task left to complete is the redesign outlined in the mockups in Figures X and Y. These designs or any additional designs should be tested with the user populations of TaleBlazer to conclude which, if any, of the changes increase the usability. The changes in styling implemented in this project are a good step towards increasing the workflow and simplicity of the TaleBlazer editor but there is still room for future improvement.

6.2 Undo and Redo

One important aspect of usability is allowing users to make errors and not have it irreparably damage the game. The TaleBlazer editor does not have the ability to undo and redo actions made in the script panel, which is a recovery usability flaw that should be remedied in the future. This undo feature would save a fixed number of actions back and the redo would save a fixed number of actions forward if the user has pressed undo. This would allow the user to delete large blocks of scripts and then recover them shortly after if they realized they had deleted something by mistake.

6.3 Inline Help

The idea of having inline help fields in the TaleBlazer editor was considered originally to be included in this iteration of the project, but priorities caused this feature to be postponed. Ideally, there would be question marks next to TaleBlazer terms that are non-standard so that the user could click on the question marks and see a small pop-up explanation of the feature. This would aid in the learnability of the product. While much of the same information is available in external tutorials, these inline help buttons would help with ease of access to the information and efficiency.

6.4 Search and Replace

Currently the new game summary feature allows the game designer access to all of the text in the game where they can subsequently use the browser's search function to find instances of a word. Ideally, in the future, TaleBlazer will have its own search and replace feature. This would make the editor more externally consistent with other code editors and would greatly improve the efficiency of changing an agent or role name in many locations across a game.

6.5 Wizards

Many times, a game designer is unclear about how to specify all of the settings when creating an agent or role, so it would be helpful to have a guide for that process. A possible improvement would be to add in "wizards" or guides that walk the designer through the different processes, such as creating an agent or creating a world. These wizards would aid in the learnability of the system if not the efficiency and would be another form of inline tutorial for inexperienced users.

Chapter 7

Conclusion

Now, after the completion of this work, the TaleBlazer editor has a more usable interface to improve upon the strength of this powerful game design tool. After careful design, testing, and implementation, the new editor is ready to be used by young adults learning TaleBlazer for the first time as well as seasoned users creating and maintaining an already large pool of games.

The efficiency of the editor has been improved with the new agent overview page, which allows the game designer to quickly modify all of their agents in one place without switching tabs as well as get a better sense of the "big picture" view of their game. The new game summary also gives the game designer a useful new tool for revising text in their games as well as a way to print out and share a game overview with others. The new error checker aids in the usability of the editor by making the game designer less error prone and able to quickly fix the common errors seen in the TaleBlazer editor. Finally, the restyling of the colors and alignment gives the editor a more polished and clean appearance, which aids in the simplicity of the interface and makes the editor appear like a modern, professional tool.

Appendix A

Poll

Poll to Assess Priorities for TaleBlazer Editor Improvements 2013-14

Here is a proposed list of changes we are contemplating for the upcoming year to make the editor easier, more reliable, and more intuitive to use. Again, these are all intended to help game designers have an easier time using TaleBlazer.

We need your help prioritizing this list so we can tackle the most desired features first. **Please take a look at the list and let me know (by number) the top 5 items you?d like tackled ASAP.

Enhanced Asset Management

1. Add royalty-free stock agent images
2. Allow users to tag/share agent images
3. Enable designers to specify Action Button images (to replace the gold star)
4. Enable designers to import/use custom map icons (not just shape/color)

Sound

5. Enable designer to include item selected from pre-set group of stock sound effects (cash register "cha-ching", "boom", "wah-wah" you loose sound, etc.)
    (a) E.g., When a player bumps a given icon, a specific sfx could trigger
6. Allow users to upload custom sound effects (e.g., a specific bird song)

7. Enable designer to include voice readings (e.g., to appear over text on screen)

8. Allow ambient sounds or music to play during game (*may not be technically feasible)

Error Checking

9. Have ability to check for errors in the game, such as:

    (a) Required but unpopulated fields

    (b) Script action with no script

    (c) Password block with no password set

    (d) Duplicate clue codes

    (e) Blocks that refer to objects that have been deleted

    (f) Unattached blocks

    (g) Checking for inaccessible agents

Content Management

10. Undo/redo actions

11. Find/Replace words, such as names, in the game

12. Spell check

13. See a game overview or summary report of all the data in a game (clue codes, agents, password protected objects, etc.)

14. Enable game designer to print a hard copy of the game

15. Allow grouping or sorting agents

Help

16. Add inline help fields (hover over '?' to provide explanatory text to help clarify terms, etc.)

17. Have “wizards” or guides that walk you through the different processes, such as creating an agent or creating a world.

18. Provide designer with an overview of agents (indicating when they dis/appear in the game, etc. without having to make them do a manual search)

Bibliography

[1] MIT Scheller Teacher Education Program. MITAR Games. http://education.mit.edu/projects/mitar-games.

[2] MIT Scheller Teacher Education Program. Starlogo TNG. http://education.mit.edu/projects/starlogo-tng.

[3] MIT Scheller Teacher Education Program. Starlogo Nova. http://education.mit.edu/starlogo-nova.

[4] MIT Lifelong Kindergarten Group. Scratch. scratch.mit.edu.

[5] Boehm B, *A Spiral Model of Software Development and Enhancement*, ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, August 1986