Usability and learnability improvements
for the TaleBlazer Game Editor

by

Jacqueline Hung
S.B., Massachusetts Institute of Technology (2014)

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
June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.
Usability and learnability improvements for the TaleBlazer Game Editor

by

Jacqueline Hung

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2015, in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

TaleBlazer is a platform for creating and playing location-based augmented-reality games. The TaleBlazer web-based editor allows users to build customized games using a blocks-based programming language. In order to allow more user groups to utilize various features offered by TaleBlazer, building an intuitive editor user interface is especially essential to maximizing usability and learnability. This thesis presents a new version of the editor with improved front-end design and a new tutorials system. The updated implementation is intuitive, powerful, and user-friendly for both novice and experienced TaleBlazer users.

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

It has been a wonderful and fulfilling experience working with the TaleBlazer team. I’d like to thank Eric Klopfer, Lisa Stump and Judy Perry for bringing me onboard. I would not have been able to work efficiently and productively without their support.

Lisa Stump has been incredibly helpful throughout the development process, and many technical issues were quickly resolved after consultation with her. I’d also like to thank Judy Perry for her guidance on project management and user testing. Only with their help could the project goal be achieved within one year.

I’d like to thank the rest of the TaleBlazer team and my predecessor Cristina Lozano. Both TaleBlazer’s back-end system and front-end interface are drastically improved by the dedication of all members on the team. It was my pleasure to work with such inspiring, patient and creative people.

I’d also like to thank my academic advisor Eric Grimson and my undergrad advisor Seth Teller who encouraged me to apply for this position.

Finally, I’d like to thank my family and friends who have always been there through every step in my life.
4.2.3 Script Blocks .............................................................................................................. 38
4.2.4 Overwrite Warning ..................................................................................................... 42
4.2.5 Miscellaneous ........................................................................................................... 45

Chapter 5 Learnability Improvements .............................................................................. 51
5.1 Background ................................................................................................................... 51
  5.1.1 Motivation ................................................................................................................ 51
  5.1.2 Review of similar platforms ....................................................................................... 51
5.2 Format & Structure ....................................................................................................... 60
  5.2.1 Display Format ....................................................................................................... 60
  5.2.2 Tutorials Structure ................................................................................................. 62
5.3 Implementation ............................................................................................................ 63
5.4 Content Creation ......................................................................................................... 67

Chapter 6 User Testing ........................................................................................................ 70

Chapter 7 Future Work ....................................................................................................... 73
  7.1 Video Tutorials ......................................................................................................... 73
  7.2 Search Function for Tutorials ..................................................................................... 73
  7.3 Interactions between the Editor and Tutorials ............................................................. 74
  7.4 Undo and Redo ........................................................................................................... 74

Chapter 8 Contribution and Conclusion .......................................................................... 75
  8.1 Contributions ............................................................................................................. 75
  8.2 Conclusion .................................................................................................................. 76

Bibliography ....................................................................................................................... 77
Chapter 1
Introduction

TaleBlazer is an augmented reality (AR) platform developed by the MIT Scheller Teacher Education Program (STEP) lab for users to create and play location-based games. As a general description, AR games allow users to walk around the real world and interact with the physical environment while following game rules and additional information on their smartphones. In TaleBlazer, AR is accomplished through a map shown on the phone corresponding to the real world environment users are walking in. On the map, there are also markers that indicate virtual agents (characters and objects) in the game. By tracking the player’s location using GPS, the TaleBlazer application will notify them if any characters or objects are nearby.

A main feature of TaleBlazer is its web-based editor that uses a blocks-based programming language to represent game mechanics. By drag-and-dropping logic blocks, even users without specialized technical know-how can easily create complex AR games without prior knowledge of programming. By arranging logic blocks and customizing game characters, users are empowered to create sophisticated game rules and various interactions with game objects. Because all the game editing can be done in a web browser, users are not required to download separate software. To complete the user web-based editing experience, all changes are saved on TaleBlazer’s server and the games can then be downloaded onto the user’s phone using TaleBlazer’s mobile application.
1.1 Motivation

TaleBlazer is an online platform aimed to enable creation of educational games with use of modern day technology, especially smartphone-based engagement. The TaleBlazer web editor has two target groups each with its own unique set of user experience design needs. The first target group is comprised of game creators at partner institutions such as zoos, schools, museums or nature centers. Such users are professional exhibit or curriculum designers who are committed to producing a polished interactive and engaging experience for their visitors. Their main purposes of creating AR games is to showcase their offerings, engage visitors with the environment, and educate people through interactive location-based games. On the other hand, the second target group is comprised of students ranging from elementary school students to high school teens. Through creating and playing educational AR games, these young users learn new subjects in a fun and engaging way.

Both of these target groups are non-technical users who have little to no experience in programming. They need the TaleBlazer editor to be as intuitive and self-explanatory as possible because neither of them have the time or wherewithal to study a complex interface. Furthermore, especially for younger users who are trying to use TaleBlazer to create games that help them learn, a clunky and unintuitive interface defeats the purpose of making education more fun and engaging. Further more, good UI tools should allow new users to get started easily (i.e. low threshold), but also allow experts to create sophisticated projects (i.e. high ceiling) with advanced functionality (Myers, 2000).

The goal of this thesis is to make significant and measurable improvements to TaleBlazer’s user experience, while lowering the threshold and enhancing the high ceiling. After conducting heuristic evaluations and user testing, I confirmed that the prior TaleBlazer web editor does not provide suitable usability for our two main user classes. To improve the editor’s usability and learnability, I redesigned the workflow as well as various user interface (UI) elements. In addition, I developed a tutorial panel to provide instant assistance.
for novel users. After completing these and a series of other improvements over the past year, I will launch a new version of the editor with the goal of creating a more elegant, powerful, and user-friendly experience for all TaleBlazer users.

1.2 Thesis Summary

Chapter 2 provides an overview of TaleBlazer including the background, design, and functions of the technology and also reviews previous work on similar platforms developed at the MIT STEP lab.

Chapter 3 outlines the problems of the prior editor design based on user testing, heuristic evaluations and survey results.

Chapter 4 presents the design process of each new UI element solution to enhance the user experience of the editor’s usability and script blocks.

Chapter 5 discusses the research behind and implementation of the new tutorials panel which hugely improved the learnability of TaleBlazer’s editor.

Chapter 6 reviews the user testing experiments and results which validate the newly improved editor.

Chapter 7 discusses future work that could be done to further improve the editor UI.

Chapter 8 concludes with thesis goals and contributions.
Chapter 2

Background

2.1 TaleBlazer

2.1.1 Overview

Throughout the past two decades, the STEP lab has been dedicated first to conducting research projects which further the field of learning science and second to applying the findings to create new sophisticated learning techniques and technology implemented through intuitive and simple formats such as educational games. TaleBlazer is one of the many technologies developed in the lab that is designed to help both teachers and students explore new ways of learning.

Driven by this mission of innovation for education, TaleBlazer is an online platform to create and play location-based AR games. AR games connect the virtual game map with the real world, typically using GPS data from mobile devices. As a user walks around in the real-world environment, the user’s smartphone detects its current location and updates the game map. Once the user gets close enough to a game character’s physical location, the phone will display the character’s description and a set of possible actions from which the user chooses how to interact with the character (See Figure 1). By following the gameplay on a smartphone, players can simultaneously enjoy the experience of interacting with real-world objects, having dialogues with game characters, and completing engaging tasks designed for TaleBlazer games.
TaleBlazer has grown to be an emerging and effective way for game designers to communicate ideas and engage players. Game designers create TaleBlazer games with maps and game rules using the web-based editor. Game players then simply download and play the games using the TaleBlazer’s mobile app. Organizations such as anyone from zoos, aquariums and museums to educational programs and after-school camps use this system to build AR games to engage visitors with their physical environment. Teachers take advantages of TaleBlazer games to encourage students to explore, imagine, and role-play. Ultimately, exploring environments and learning new subjects are now both more engaging with TaleBlazer.

2.1.2 Editor

TaleBlazer’s website includes basic information about the project and our partners, support documentation, and most importantly, the TaleBlazer editor. Users can log in to the TaleBlazer system and see a list of any games they created as well as public games shared by other users.

The most critical component of the website is the TaleBlazer editor – where all games are created. In order to build TaleBlazer games using our editor, game creators have to first understand the basic mechanism of TaleBlazer AR games. For each TaleBlazer game, the game designer selects a map that reflects the real-world region where the game will take place. The designer can then add a game character or an item to a specific location on the map, and these characters and items are called “agents” in TaleBlazer terminology. “Traits” are variables specified for agents, for player roles, or even for the game world. “Actions” can be text, video or scripts such as “pick-up” or “drop” (See Figure 1). Once the agent is placed on the map and the game is begun, a set of “traits” and “actions” associated with each agent will show up as a dashboard on the player’s phone when the GPS signal is close enough to the agent’s location. We refer to this process of encountering an agent as “bumping”. Once the player bumps an agent, this agent’s “traits” will be displayed on the
phone (e.g. score: 95) while a list of available actions will be shown as buttons from which the player can select.

Figure 1: Yellow Scrap of Paper is an agent with an action button “Pick Up”

The prior implementation of the TaleBlazer editor consists of four tabs – World, Map, Agents, and Player. Except for the Map tab, all tabs share a similar layout with a properties panel on the left hand side and a script panel on the right. Please see the figure below for an illustration of the prior implementation.

The first tab “World” includes all the settings that are relevant to the entire game world in the properties panel on the left and the scripts for the game world on the right (See Figure 1). The properties panel allows designers to:

1. configure the mobile tabs that will display on the game player’s mobile app
2. choose which soft keyboard should be presented to players
3. add an introduction and multiple scenarios for the game
The second tab “Map” allows designers to specify where the game will take place. The “scroller ribbon” on the top shows a list of regions in tiles with the selected one in orange (See Figure 2). Each “region” can either have a Google map image or a customized image as the background. Designers can designate a game map by either entering latitude and longitude boundaries or (more commonly) by inputting an address into the embedded Google map engine.
The third tab “Agents” is where designers create game characters and items. Similar to the Map tab, there is a “ribbon” on the top showing a list of agents. Once the designer selects an agent to modify in the ribbon, she can assign it a name, image, description, and various other traits and actions in the properties panel. On the right, there is the script page for each agent that determines its behaviors. The mechanics of script blocks will be explained in detail later in this chapter.

The fourth tab “Player” shares a very similar layout with the Agents tab, *i.e.*, a ribbon on the top, properties panel on the left and scripts area on the right. This tab is where game designers choose how they want to configure player roles. If there is only one “player role”, all game players will experience the same game by default. However, designers can add multiple “player roles” to incorporate different game dynamics based the selected “role.”

### 2.1.3 Script Blocks

The ScriptBlocks library used in the editor was developed by members of the STEP lab. Users specify game rules by dragging and dropping script blocks on the script page instead of typing code (See Figure 4).

![Figure 3: Dragging script blocks onto the script page](image)

In TaleBlazer, these logic-blocks fall into six main categories – control, operators, game, looks, movement and traits. By modifying parameters and connecting various kinds of
blocks in various orders, users can construct the exact game logic they desire to create. Without needing to learn any computer language syntax, non-technical users are enabled to build mobile games by simply combining script blocks. Thus, the major advantage of using this block-based language is its ability to provide a user-friendly platform for game designers who lack programming expertise to create AR games with sophisticated gameplay.

Figure 4: An example of script blocks

Figure 5 is an example of how to connect various script blocks to create sophisticated gameplay, in this case an interaction with a ghost named Robin in a haunted house game. When the player gets close enough to the agent named Robin (i.e. when the player bumps Robin), the description of Robin will show up on the player’s phone as “Did you bring me what I asked for?” Meeting Robin will then increase the player’s “visited ghosts” variable by 1. The action “Go to inventory” will then show up as a button for the player to click on. If the player has visited more than five ghosts, then “Unlocked chest” will be added to the game world. Every step in this example of script-blocks based gameplay is straightforward and does not require any formal programming experience to comprehend.
2.1.4 Technologies

The TaleBlazer editor’s front-end is written in JavaScript. The code is then minified using Closure compiler, which makes JavaScript run faster and provides type checking. In addition to the editor, TaleBlazer has four other software components – a mobile application used to play the games, a repository server for storing and communicating with mobile devices, an analytics server, and a multi-player server under development.

All game content, images and videos are stored on the server when a game designer saves his game. When a player downloads this game on her mobile device, all the relevant files will be sent from the server for the player to begin the game. TaleBlazer’s mobile app is built using Titanium Appcelerator and is available for both iOS and Android systems. Titanium is a cross-platform JavaScript framework to develop apps on iOS and Android systems. The TaleBlazer app has a set of tabs across the top (See Figure 6) to help players navigate between the map, player dashboard and inventory. These mobile tabs can be configured by the game designer in the TaleBlazer editor.

Figure 5: Mobile tabs on the top can be configured in the web editor
2.2 Similar platforms

There are many research groups at MIT that are dedicated to inventing new technologies with the goal of making learning and programming more intuitive and interactive. Educational software developed in the MIT Media Lab includes Scratch, StarLogo, App Inventor and GameBlox. Below I will review these platforms that target novice programmers and discuss their approaches to similar UI challenges that TaleBlazer faces.

2.2.1 MITAR

TaleBlazer can be considered the descendant of MITAR, a previous software platform developed in the STEP lab to create AR games. Similar to current TaleBlazer games, MITAR allowed players to walk around the real world and interact with game characters based on their location. Even though the basic game mechanics have not changed much since MITAR, there have been major improvements to the game development process and system infrastructure in TaleBlazer. MITAR’s editor was a windows application that required separate installations and complicated processes to deploy games. TaleBlazer’s editor is a simple web-based application that runs cross-platform. In addition to the upgrade in the software architecture, TaleBlazer also supports blocks-based programming where users simply drag and drop scripts to build game rules. In MITAR, users were required to complete a long and tedious process of filling out templates. Since MITAR did not support scripting, the gameplay was limited and less engaging. TaleBlazer was designed and built to move towards a web-based editor with games stored and retrieved from a central server. The inclusion of a programming language also drastically increases the configurability of the game mechanics.
2.2.2 Scratch

Scratch is a project developed by the Lifelong Kindergarten (LLK) Group at the MIT Media Lab. Users can create interactive animations with Scratch’s blocks-based programming system in a web-based editor. The goal of Scratch is to help young people learn how to adopt a systematic approach to programming while remaining creative and imaginative with their content. Scratch allows users to make, play, and share single player interactive stories or simple games within a single web application. Scratch games do not utilize location-based technologies like TaleBlazer games, but they have many useful features for game designers to create interactive videos and work collaboratively with others in an online community. Scratch also provides very comprehensive tutorials, a forum for users to discuss ongoing projects and a user-friendly editor that is used by thousands of users every month.

2.2.3 GameBlox

GameBlox is another game editor developed by the STEP lab for users to create video games using a blocks-based programming language. It is currently primarily used in an edX course called “Design and Development of Games for Learning” and a course at MIT called “11.126x Introduction to Game Design”. GameBlox allows users to make and play multi-player games within a single web application with a much higher emphasis on games specifically. GameBlox’s editor allows game designers to create game components using “sprites,” design a game the background, and set properties for each game stage.

2.2.4 StarLogo Nova

StarLogo is an agent-based simulation software for users to model decentralized systems and create animations using the Logo programming language (with lower emphasis on games). It is another project developed by the STEP lab to lower the learning barrier for
programming. StarLogo Nova is a new branch of StarLogo introducing better 3D graphics and a blocks-based programming user interface. The ScriptBlocks library used in StarLogo Nova is shared with TaleBlazer’s editor. Users can simulate complex systems and create video games with rich visual components using StarLogo’s 3D graphics and sounds effects.

2.2.5 App Inventor

App Inventor is an open-source web platform developed by Google and MIT. It is widely used around the world to help beginners build Android apps and also utilizes blocks-based programming like TaleBlazer, Scratch, and StarLogo. App Inventor allows users to first add components, such as buttons, and then program the logic using blocks. It then compiles the code and runs the app on a connected Android device for immediate testing and troubleshooting during development. More than 7 million Android apps have been developed by members of the App Inventor community. App Inventor provides newcomers ample introductory resources and online support from its roughly 3 million users worldwide.
Chapter 3
TaleBlazer Editor UI Evaluation

The web editor is one of the most essential components of TaleBlazer because its capacity determines what types of AR games can be made and played on TaleBlazer. Over the years, the TaleBlazer team has been constantly improving the system and adding new features. The infrastructure is robust and powerful, which allows designers to create complex game mechanics and have full control over many game settings. In comparison to the strong backend functionality, the TaleBlazer editor’s frontend seems a bit behind in terms of its level of development and sophistication.

There are some differences in behaviors of the two target groups which make it more challenging to design an interface that performs well simultaneously for both of our target user groups: First, many adult game creators (e.g., teachers or education staff at a zoo) tend to use TaleBlazer regularly to create games for different occasions. Because of their regular use, they are generally more familiar with TaleBlazer terminologies and the editor interface. To these users, the efficiency and workflow of the game creation process is particularly essential. They also appreciate more advanced and customizable features in order to design more games that are more complex and sophisticated. On the contrary, many students only have a one-time experience with TaleBlazer such as during a summer camp or for use in a specific class. For these novel users, instead of optimizing customizability or efficiency, we need to minimize entry barriers that would prevent them from creating their first game.
3.1 Heuristic evaluation

To begin the process of redesigning the prior TaleBlazer editor interface to provide a better user experience, I completed a heuristic evaluation using the following heuristics (Nielsen, 1993):

- Heuristic 1: Visibility of system status
- Heuristic 2: Match between system and the real world
- Heuristic 3: User control and freedom
- Heuristic 4: Consistency and standards
- Heuristic 5: Error prevention
- Heuristic 6: Recognition rather than recall
- Heuristic 7: Flexibility and efficiency of use
- Heuristic 8: Aesthetic and minimalist design
- Heuristic 9: Help users recognize, diagnose, and recover from errors
- Heuristic 10: Help and documentation

This heuristic evaluation resulted in the following eight problems with the prior TaleBlazer editor:

1. **Problem**: The World tab is the first display a new user sees when she clicks into the editor. Yet, no special tutorials or messages are displayed to assist new users with getting familiar with the development environment despite the fact that neither of our user classes has sufficient technical background in programming and AR game development. There are also many terminologies (agent, traits, region, etc.) that most new users are not familiar with but are constantly used in the editor without explanation. Although there is an existing PDF document for reference, separate tutorials and documentation are generally not helpful in the development process and increase the initial learning barrier.
Heuristics: 10
Severity: Serious

2. **Problem**: In the script editor, it is especially hard for novice users to find a specific block without memorizing which category it is under. Without any documentation or inline help, it is challenging for beginners to get started and learn how to connect different types of blocks.

   Heuristics: 10
   Severity: Serious

3. **Problem**: The current process of reverting back to an earlier revision of their game is clunky and requires improvement. Moreover, multiple people could be editing the same game and there is no warning when a game is opened on a different machine or in another tab. As a result, users occasionally accidentally overwrite each other’s changes, which causes a great deal of frustration.

   Heuristics: 5, 9
   Severity: Serious

4. **Problem**: It is not obvious to users that their games are not saved unless they click on the save button. Since TaleBlazer does not save games automatically, users can lose the games that took them hours to create if they forget to save.

   Heuristics: 5, 1
   Severity: Minor

5. **Problem**: Buttons have different styling formats and do not take advantage of common icons. The layout seems crowded on certain parts of the page yet there are many unnecessary blank spaces throughout the editor as shown in Figure 7.
6. **Problem:** It is not clear that users have to drag and drop scripts to the trash can icon, instead of clicking on it.

   **Heuristics:** 4, 8

   **Severity:** Minor

7. **Problem:** There is no easy way to access the “My Games” page with a list of games created by the user. This problem adds confusion for new users because there is important information such as the game code located on the “My Games” page.

   **Heuristics:** 7, 10

   **Severity:** Minor

8. **Problem:** The editor does not support undo and redo.

   **Heuristics:** 3, 7

   **Severity:** Minor
3.2 User testing

To better prioritize the list of feature requests, I conducted interviews with five TaleBlazer users. Two of them are beginners to TaleBlazer and the other three are experienced editor users. Two of these experienced users are working in our partner institutions, and so their interviews were conducted via online screen sharing. The remainder of the interviews were conducted in person using the users’ laptops. Interview lengths ranged from 40 minutes to an hour and the screen interactions were recorded using QuickTime (for review and analysis following the interview).

The interview structure is as follows:

How experienced are you with TaleBlazer editor?
   a. 0: No experience - Haven't used it before
   b. 1: Little experience - Only used it once
   c. 2: Pretty experienced - Created less than five games
   d. 3: Very experienced - Created more than ten games

2. Are there any problems/questions you or other users face when creating games on TaleBlazer’s editor?

3. Which tabs do you find confusing or hard to use and what about that tab is confusing for you?

4. Can you create a simple game with three agents?

5. Can you add some actions and traits?

6. Can you add some scripts for your agents?

7. How helpful would it be if we add:
   a. A searchable help page
   b. inline or pop-up tutorial
   c. undo / redo functionality
   d. improve the assets (icons, images, etc.)
### 3.2.1 Results

Below is a list of problems including direct user feedback as well as issues I identified after reviewing each user’s interaction with the editor:

1. The World tab should not be the first page shown to users. It is not very informative and is not essential in the process of creating a simple functional game.
2. There are some major trackpad gesture conflicts. For example, on the Agent tab, scrolling to the left makes the browser go back a page without saving. On the Map tab, scrolling up zooms in the map when the cursor is over the map area.
3. The navigation system is clunky. There should be a link to “my games” and to the game code in the editor.
4. Script Block category names could be confusing (Game, Looks, etc.) Common mistakes include, erroneously looking for the “switch to” block in the “Movement” category, unsuccessfully searching for the “Inventory” Block, and unsuccessfully searching for the “action” block.
5. The map is locked by default so it is difficult for new users to find the address bar. Also, sometimes users forget to press “Move map to here”.
6. The trash and script drawer should be sticky (*i.e.* stick on the top of the screen when scrolling); otherwise it is difficult to scroll up for long scripts.
7. Newly added action scripts sometimes show up beneath the trash icon or off the screen.
8. Drag to select does not include long scripts that are off the screen.
9. The demo game does not have any scripts in the World tab, and no traits were added to the agents. The demo game should be updated to educate users about the game world, actions and traits.
10. The Game dashboard does not collapse automatically.
11. It is not obvious to novice users what adding multiple roles entails.
12. It is not clear to new users how to add variables for scores.
13. Users want to see what tabs look like in mobile (Mobile Emulator). Perhaps the editor can include mobile icons.
14. Agents all initialize at the same place on the map.
15. Scripts cannot be connected from the top.
16. Users cannot reuse the same scripts in different games.
17. Users cannot add global scripts \(i.e.\) scripts applying to all agents.
18. When filling out input fields for Say or Set scripts, the cursor will jump to the end upon key press, making it impossible to edit text.

Below are users’ aggregated responses to question 7 – which new feature would be the most helpful:

a. A searchable help page might be helpful for adult users, but it’s unlikely that kids would want to read instructions before starting.
b. An inline or pop-up tutorial can be very helpful for new users.
c. Undo/Redo functionality is relatively less important because any big mistakes could be recovered by reverting back to an older version. However, Undo/Redo functionality could be particularly helpful to un-trash the most recently trashed script block.
d. Improvements on assets and stock images are not as helpful for organizations that would be using their own images, but it might be useful for kids to have access to stock images for common agents/objects.

3.2.2 Conclusion

Through observation of both novice and experienced users interacting with the TaleBlazer editor, I gained valuable insights into the core usability issues with the prior editor’s UI design. The most common requests are related to redesigning the panels and script drawer to
improve learnability and avoid confusion. These changes would involve creating new UI elements and restructuring script blocks. Regarding new features requests, interviewees all agreed that inline or pop-up tutorials would be the most helpful.
Chapter 4
Usability Improvements

4.1 Design mockup

In order to create new designs that reflect users’ game creation workflow, we have to first understand the essential steps to creating games using the TaleBlazer editor. First, users can access the editor by creating a new game or clicking the ‘Edit Game’ button on one of their existing games on the “My Games” page (See Figure 8). This page also includes some important information, such as the game code, attribute settings, public/private settings, and restore options.

![Figure 7: The “My Game” page with game code and other information about the game](image)

The prior editor does not provide an easy way to get back to the My Games page. The only way is to click the browser back button after saving the game because otherwise users will be prompted to save before leaving the editor page.
Figure 8: The original Save button is visually similar to the New Game button

Many users also complained that the New Game button, which is right next to the Save button, is so similar in terms of raw visual styling that people have accidentally clicked on New Game when they intended to save (See Figure 9).

To provide users a better navigation experience between the TaleBlazer website and the editor, I introduced a navbar on the top. This navbar will not only take advantage of the currently wasted empty space but will also incorporate two previously implemented features: “Download Summary” and “Error Check”. The Download Summary button exports an HTML file with important information about the world, agents and players. The Error Check button toggles a panel from below to display a list of warnings and errors (e.g. missing arguments, empty script/text/video action).

Another important topic related to navigation is how to achieve a more natural workflow with a better organization of the tabs. The original editor had 4 tabs in the following order – World, Map, Agents, Player. The first tab that users would see was the World tab as shown in figure X. During user testing as described in chapter 3, I found that the World tab has systematically been the most confusing tab to the vast majority of users. From observing how users create games, I found that setting up a map and adding agents to the map is the most common approach. Thus, displaying the World tab first does not reflect the usual workflow. Another reason why this tab is particularly confusing for users is that it includes some mobile and properties settings, which is not directly related to the game world. Since we have identified the most common workflow through user testing and interviews with experienced users, I reorganized the tabs to better reflect the common and most intuitive game creation process. The new tab system includes the reordered and redesigned Map,
Agents, Player, World tabs with an additional tab called “Settings”. All the properties-setting panels that are irrelevant to the game world are migrated from the World tab to the new Settings tab. Map and Agents become the first two tabs as they are usually the first two steps of creating a TaleBlazer game.

One critical usability issue based on user feedback as discussed in chapter 3 is that some built-in trackpad gestures on Macintosh computers conflicts with certain parts of TaleBlazer UI, and in some cases these conflicts would cause users to accidentally lose all their edits. This often occurs when Mac users scroll left using their trackpads, which triggers a browser event to go back a page and so brings the user back to the My Games page. Unfortunately, as the earlier version of the TaleBlazer editor utilized horizontal scrolling in all scroller ribbons (Map, Agents, Player) as shown in Figure 10, it is almost impossible for mac users not to accidentally scroll left, especially as more agents are added to the game. Based on my interviews with a few TaleBlazer partners who had significant experience helping younger students acclimate to TaleBlazer, many teenage users have suffered from this problem and lost all of their changes when scrolling through the agents.

![Figure 9: The original agents ribbon with horizontal scrolling](image)

In order to solve this critical issue, a pop-up dialogue is added to prompt users to save their game before going back a page. In addition, to reduce the frequency of left-scrolling, I also changed all scroller ribbons to vertical scrolling (See Figure 11).
4.2 Implementation

The editor frontend is implemented in JavaScript and Google Closure. Its backend is written in CakePHP which follows the model–view–controller (MVC) architectural pattern. A controller sends commands to update the model. The model notifies the view of its change of state. The view uses this information to output the corresponding representation to users. These view files are written in PHP with an extension of .ctp (CakePHP Template). They include logics to convert data received from the controller into a presentational format to be displayed on the webpage.

Our system follows the observer pattern, where an event manager keeps track of event handlers and notifies its “observers” automatically of any state changes. When a certain event is triggered (an agent is deleted from the Agent tab for example), the event manager
will call the designated method (dispatch Delete event in this example) and updates all the other UI views.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 problem 1, 3.2.1 problem 1</td>
<td>4.1 Redesign, 5.3 Tutorials</td>
</tr>
<tr>
<td>3.1 problem 2</td>
<td>4.2.3 Script Blocks, 5.3 Tutorials</td>
</tr>
<tr>
<td>3.1 problem 3</td>
<td>4.2.4 Overwrite Warning</td>
</tr>
<tr>
<td>3.1 problem 4</td>
<td>4.2.1 Navigation</td>
</tr>
<tr>
<td>3.1 problem 5</td>
<td>4.1 Redesign, 4.2.5 Layout and Styling</td>
</tr>
<tr>
<td>3.1 problem 6, 3.2.1 problem 4</td>
<td>4.2.3 Script Blocks</td>
</tr>
<tr>
<td>3.1 problem 7, 3.2.1 problem 3</td>
<td>4.1 Redesign, 4.2.1 Navigation</td>
</tr>
<tr>
<td>3.2.1 problem 2</td>
<td>4.2.2 Scroller Ribbon</td>
</tr>
<tr>
<td>3.2.1 problem 5</td>
<td>4.2.4 Map</td>
</tr>
<tr>
<td>3.2.1 problem 6</td>
<td>4.2.3 Script Blocks, Toolbar</td>
</tr>
<tr>
<td>3.2.1 problem 10</td>
<td>4.2.1 Navigation</td>
</tr>
</tbody>
</table>

Table 11: List of problems in Chapter 3 and its corresponding solutions in Chapter 4 and 5

4.2.1 Navigation

The new navbar is added to the default editor layout so it would always be displayed on the top of the editor (See Figure 12). As icons are helpful visual cues, especially for a non-technical audience, we designed appropriate icons for the navbar buttons. I arranged “My Game” and “New Game” on the left – the two buttons that will bring users to a new page. To enforce consistency, we also decided to navigate to the linked URLs (*i.e.*, http://taleblazer.org/profile/mygames/ & http://taleblazer.org/create/) via a new tab so users can not only keep the current game but also quickly access another page without being prompted to save or discard changes.
The 4 buttons arranged on the right are Tutorials, Summary, Error Check and Save Game. All of these are specific to the current game and will not bring users to another page. Clicking the summary button will trigger a download of the game summary HTML file. The Tutorials button toggles a panel from the right, and similarly, the Error Check button toggles a panel from the bottom. Chapter 5 will explain the addition of the tutorials in further detail. The Save Game button has two states, depending on whether there has been any changes since the game was last saved. If the game file is in the “dirty” state (i.e. there are updates that have not yet been saved), a red dashed border around the Save button will fade in as a reminder for game designers to save the game (See Figure 13).

The original top bar has a small game thumbnail and the game name on the right. I removed the small game thumbnail because it is too small to be meaningful and the default image (question mark) only causes confusion for novice users, who erroneously interpreted the icon as ‘help’. For the game name input field, I handled text overflow by adding ellipsis to the end for long names. Users can choose a game image and add a description by toggling the game dashboard.

The original game dashboard does not collapse unless the user explicitly clicks on the accordion collapse (triangular) icon on the far right. Both new and experienced users find this very inconvenient. I added a handler that listens to any click events that happens outside of the expanded game dashboard, and automatically collapse the dashboard once users click elsewhere. The game code is also added to in the dashboard (See Figure 14), so users will no
longer need to go back to the My Games page for game codes. This popular feature had been requested by multiple TaleBlazer partners.

Figure 14: The new game dashboard with game code

The redesigned tab system also adopts a more modern-looking tab styling with slanted edges like Google Chrome or FireFox tabs (See Figure 15). The font and paddings are also adjusted to remove inconsistency in font sizes and unnecessary spacing.

Figure 15: TaleBlazer’s tab design changed from the above to the redesigned look below
4.2.2 Scroller Ribbon

To avoid the trackpad gesture conflicts, we changed the scroller ribbon from horizontal scrolling to vertical scrolling. Making this specific switch is technically trivial, but we wanted to do so after careful consideration. First of all, the ribbon has a relatively small height. As the scrollbar thumbs reduces in size to accommodate new agents as they are added, the small height can cause overly small agent icons if the agent number is above even a modest amount. We also considered the option of making the scroller ribbon vertical as shown in Figure 16 and 17. However, we eventually found that this layout does not solve the problem of scrollbar thumbs excessively shrinking when there are many agents. Instead, it takes away horizontal space and might compromise the user experience with long script blocks.

Figure 16: Mockup idea 1 with vertical scrolling
By interviewing TaleBlazer users and partners, we learned that game designers primarily use the ribbon in two distinct situations. When they are creating agent scripts, game designers want the ribbon to take less space. Conversely, when they try to find a specific agent from the ribbon, they want it to be bigger to show more agents at a time and thereby reduce search time.

Based on this finding, I added an “expand all” functionality to show all the agents in one view (See Figure 18 and 19). I also implemented an auto-focus function which listens to the event of a tile being clicked on and then automatically scrolls the ribbon container so that the selected tile can be displayed in full. The New Agent button is also moved to the empty space on the top right.
Modern UI design exploits white spaces and different shades of background color to define layout. The original TaleBlazer editor used borders to outline every container which makes the UI seem crowded and archaic. Based on contemporary UI design best practices, I removed the border lines, defined the container using a light grey background color, and added padding between tiles. Within each tile, I adjusted the padding and handled long agent names with ellipses. The original tiles have all trash icons in red for the purpose of alerting users of a potential delete action. However, the red icons seem distracting and excessive when combined with a safety prompt requiring users to confirm any delete operations after clicking. To minimize distractions and maintain a clean interface, I changed the styling of the trash icon so that the red background will only fade in on a mouse-over event (See Figure 20).

Figure 20: The original agent tile design (left) and redesigned tile (right)
4.2.3 Script Blocks

The ScriptBlocks library is a product of collaboration between members of StarLogo and TaleBlazer. Although most of the backend code is shared between the two, customizations are often added separately to fit different needs from each team. On the top of the drawer container, there is a simple select menu with drawer categories. There are six categories in TaleBlazer: control, operators, game, looks, movement and traits. Once the category is chosen, its associated scripts will show up in the drawer container below. Game designers drag a script from the drawer and drop it on the scripts page (See Figure 21).

![Figure 21: The original script page UI](image)

Scroll-through Drawers

The two step process – choosing category first and then the block – can be very inconvenient for novice users who are not familiar with the scripts categorization. During user testing, I realized that it became a trial-and-error process for users to find a specific script. For example, I recorded the process of how one of TaleBlazer’s new users found the “inventory” block. He first clicked on the category “Game” and scrolled down to look for
inventory. After realizing that inventory is not in this drawer, he then repeated this process three times, looking through different categories (“Traits”, “Looks” and “Movement”). He finally found the inventory block in “Movement” after 4 scrolls and 8 clicks (1 click on dropdown plus 1 click on menu select).

I reached out to the StarLogo team and we decided to collaborate on this project since it would be very beneficial for all TaleBlazer and StarLogo users. To remove the disconnection between drawers, we decided to make the drawer container scrollable so that users can scroll from one drawer to the next without going through the select menu. I integrated the code written by Polly Lal, a member of the StarLogo’s team, so that her JavaScript implementation would be compatible with TaleBlazer’s structure.

To build a scrollable drawer, we append each drawer along with its blocks to one long list. We keep track of the index of each drawer and automatically scroll the container to the drawer selected from the menu. With this approach, users can scroll through the drawers easily and quickly find the script they are looking for. Horizontal scrolling appears when we detect any scripts in view that is longer than the width of the container. To provide even more convenience for users to see long scripts easily in one view, I also made the drawer container resizable. A resizing handle (arrow icon) will show up when the cursor approaches the edge of the drawer container. I also wrote a handler that adjusts the size of the scripts page appropriately after users resize the drawer. By resizing the drawer to user’s desired width, game designers can now customize their script editor environment to fit their preferences (See Figure 22).
Figure 22: The redesigned script page UI with a redesigned toolbar and scrollable drawers (The drawer container can be resized by moving the arrow handle)

**Toolbar**

The ScriptBlocks library supports copy, cut, paste and delete operations. It also allows users to toggle between drag to scroll and drag to select (See Figure 23).

Figure 23: The original toolbar and trash UI
During user testing, I found that it takes new users a while to learn that the trash icon shown on the right is actually not a button. A block must be dragged onto the trash icon to be deleted. Such a UI element is not consistent with common visual cues for drag-and-drop operations. One easy solution is to provide some text description and a box with dotted outline, that suggests a drag-and-drop action. Also, I see no reason why the trash has an icon, but the others (copy, cut, paste) are all in default button format. Therefore, I redesigned the tool bar to give it a fresh look with matching icons. When a script is dragged to the trash area, the trash container will light up in red to obtain users’ attention (See Figure 24).

Figure 24: The redesigned toolbar UI (left) and the dragging state of the trash (right)

Additionally, another usability problem raised by our partner institutions occurs when game designers are forced to constantly scroll up to use the toolbar, which can be a nuisance especially with lengthy script block chains. The original toolbar is on the top of the scripts page and is not easily accessible if the scripts go over browser screen height. To provide users a more convenient and efficient interface, I made the toolbar “sticky” – so that it would always stay at the top right corner no matter how far down the user has scrolled. Initially, I tried to implement this by adding a CSS class that set the toolbar to fixed positioning once a user has scrolled past the top of the script page. However, this caused the toolbar to flicker during scrolling. It turns out that in some browsers this fixed-positioned element will be redrawn every time a user scrolls and there has not been a reliable cross-browser solution. As one way to work around this problem, I created a duplicate of the toolbar with all the event handlers copied over. This toolbar copy has fixed positioning in the top right corner and will only be visible once a user has scrolled past the original toolbar.
As a result, this solution achieves the same effect of making the toolbar “sticky” and easily accessible when users are working on agents with hundreds of lines of scripts (See Figure 25).

![Figure 25: The redesigned “sticky” toolbar that stays on the top right after scrolling](image)

### 4.2.4 Overwrite Warning

In chapter 3, we learned from our interviews with experienced TaleBlazer users that it is possible that more than two users are editing the game file at the same time. This issue can also occur when a single user has the same game open in two tabs or browser windows. The original TaleBlazer does not provide any warnings when this happens, and multiple users might be overwriting each other’s game file. This is a particularly serious problem for staff members at partner institutions (e.g. zoos, botanical gardens, and museums) who are responsible for working on huge game collaboratively with colleagues. For example, a game
designer Alice opened her game at 1pm and started to design 5 new pirate agents. Her colleague Bob does not know Alice was working on the game. He opened the game at 1:30pm, before Alice had saved, and added a new Treasure Island map. Alice finished her pirate agent and saved the game at 2pm. Bob saved his version with the new map at 2:30pm, without knowing that this will overwrite the changes Alice just made. The next time Alice opens the game, she would be very frustrated to find that the game has Bob’s new Treasure Island map but none of her pirate agents.

If TaleBlazer were a single offline application, having the last save automatically overwrite the file would not have been a problem. However, we need to handle the saving situation with more caution because TaleBlazer is a collaborative online platform. There are multiple approaches to solve synchronization problems like this saving problem. Obviously, if we have no resource limits on developers and hardware, building a responsive editor which multiple users can interact with at the same time like Google Docs would be one solution. However, this route would require too much time and development resources, and it seems like overkill considering the scale and use cases of TaleBlazer. Google Docs is designed to allow many people to edit a single file simultaneously as a primary feature of the product while it is very uncommon for more than 3 people to be working on the same TaleBlazer game.

Another easier and faster solution is to run a quick check before executing a save command. Using the same example given above, when Alice clicks save, the editor would query the server and confirm that nobody has saved another version to this game since Alice opened the game. With this new checking mechanism, Bob will get a warning when he clicks save. Bob can choose to proceed (save his version and overwrite Alice’s changes) or cancel the save operation. Upon getting this warning, Bob could consult Alice first and would no longer accidentally overwrite Alice’s changes.

One problem with this ‘checking before saving’ approach is that a user could have spent hours making edits without saving in between (even though this is obviously against
TaleBlazer best practices). Under these circumstances, it would be a difficult decision to choose between overwriting other people’s changes and discarding one’s own updates. Therefore, I decided to run this check every five minutes – a time period that is long enough to not be so excessive that it might slow down the GUI, but short enough to ensure the user hasn’t unwittingly made too many changes that may have to later be discarded.

Figure 26: A warning pops up when this game is opened elsewhere

I wrote the update-checking function using the JavaScript setInterval method which runs a code snippet repeatedly with a fixed time delay between each execution. Every five minutes, the editor queries the server to check if the latest version number saved on file has been changed. If so, an alert as shown in Figure 26 will pop up.
4.2.5 Miscellaneous

Properties Panel

In order to offer the most horizontal space for long scripts, the properties settings panel on the left needs to be collapsed. The original design has a big toggle button on the right. When you click on it, the properties settings will be hidden, leaving only the toggle button on the screen (See Figure 27).

![Figure 27: The original properties panel when expanded (left) and collapsed (right)](image)

The original toggle button was a bit too big and the collapsed version shows zero information about the properties panels. I researched modern web interfaces and editors with collapsible sidebars and borrowed a design concept used by Photoshop and Asana (See Figure 28).

I moved the toggle button to the top to avoid wasting horizontal space, similar to the Photoshop collapse button. Instead of hiding the whole side panel, I reduced the width so that a narrow column serves as a preview of the collapsed panel. I wrote an event handler for the top toggle bar that animates the transition of a full-width panel to a narrow column, and the panel eventually dims out as its transparency value increases throughout the sliding process (See Figure 29).
Figure 28: Asana sidebar expanded (left) and collapsed (middle)

Figure 29: The redesigned properties panel when expanded (left) and collapsed (right)
Map

In the Map tab, users can choose a map by specifying the longitude/latitude or entering an address. However, the address search bar is not visible when the map is locked in the prior implementation. Thus, many novice users are not aware that they can input an address to determine game location (See Figure 30).

![Figure 30: The original map tab UI with no guidance on the process of setting up a map](image)

After a user has unchecked “Lock Map”, the address search bar will appear. Users can type in an address and hit the “Search Address” button for the Google map to find this location. However, we found during user testing that many users forgot to press the “Move Game To Here” button in order to move all agents to this new location (See Figure 31).
As a solution to the two problems above, I made two changes. First, I made the address feature visible but disabled when “Lock Map” is checked. I greyed out the area of the disabled buttons by simply placing a half-transparent div with a higher z-index. In the address input field, I also provided a hint saying, “Unlock map to search for an address” to guide new users through the map creation process. Once a user unlocks the map, this cover div will be hidden and users are then allowed to type in an address. Once the user clicks on the search icon button or hits the enter key, the map will switch to the new location and the “Move Game To Here” will then appear prominently to remind users to move all game agents to the newly chosen map (See Figure 32).

Figure 32: The redesigned address bar in 3 states: map locked (top), mop unlocked (middle), address entered (bottom)
Styling

The original editor does not have a consistent styling or design theme across the site (See Figure 33). The buttons are in different formats and many other UI elements are using default browser styling. The CSS structure was also very inconsistent and violated many modern CSS architectural best practices. Some CSS code was defined for a single element, which did not provide scalability and modularity. I restructured the editor CSS so that styling specifications are in CSS classes instead of being redefined on every single element. I also added CSS classes for icon buttons and designed all the icons together with a consistent appearance (See Figure 34 and 35).

Figure 33: The original editor UI
Figure 34: The redesigned editor UI

Figure 35: The original tooltip (above) and the redesigned look (below)
Chapter 5
Learnability Improvements

5.1 Background

5.1.1 Motivation

The goal of my project is to create an intuitive design for the TaleBlazer web editor. I want to lower the entry threshold while maintaining a high ceiling, meaning that my redesigned user interface should be very intuitive and simple to learn for novel users, but all the complex functionalities are preserved, if not enhanced.

During our initial user testing, 2 out of 3 novice users were not able to complete their first TaleBlazer game without me providing further explanation. Moreover, the tutorial documents for TaleBlazer were not accessible from the editor page. Being able to receive help during the development process is essential to the learnability of an editor interface with such complexity. Moreover, it is also very important to integrate the documentation with actual practice. Most users do not read long PDF documents before they start using a platform. During user testing, all users agreed that it would be extremely helpful to build pop-up tutorials and inline help to guide new users through the basic steps and acquaint them with the tools we provide.

5.1.2 Review of similar platforms

In order to learn more about the work flow for script block languages and intuitive online editors, I researched similar editor platforms, such as Scratch, App Inventor, and Stencyl,
etc. Each of them has various approaches to engage new users, but all of these editors share the same principles – structure, simplicity and visibility. Structure is good organization and placement of UI elements. Simplicity allows users to seamlessly interact with the UI and access features. Visibility refers to the minimization of clutter and distraction on the screen.

**Scratch -** [http://scratch.mit.edu/](http://scratch.mit.edu/)

Scratch’s approach to improve learnability:

1. Intro video on home page

![Figure 36](image)

2. Featured work and examples on home page for users to preview the end product produced by this platform

![Figure 37](image)
3. Step-by-step intro in a side panel

Figure 38

Clear instructions accompanied by screenshots and animation

Figure 39
4. Editor map pops up when users click on help icon.

Figure 40

5. Associate scripts with different colors

Figure 41
6. Make use of icons that are consistent with other interfaces

Figure 42

7. Dedicated help page

Figure 43

8. FAQ and forums

Figure 44
9. Video tutorials

App Inventor - http://appinventor.mit.edu/explore/

App Inventor’s approach to improve learnability:

1. Screenshot explanation

Figure 45

Figure 46
2. Links to get-started intro and tutorials on the homepage

   ![Image 119x429 to 515x699]

   Figure 47

3. Beginner video tutorials

   ![Image 119x284 to 491x360]

   Figure 48

4. Tutorial Filter

   ![Image 119x144 to 493x231]

   Figure 49

5. Designate difficulty levels to tutorials

   ![Image 119x429 to 515x699]

   Figure 50
6. Initial pop-up for new users accessing the editor

![Welcome to App Inventor 2!]

*Figure 51*

7. Troubleshooting page with common issues

<table>
<thead>
<tr>
<th>Quick Links to Common Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updating Setup Software</td>
</tr>
<tr>
<td>Build Apps &amp; Share (Google Play)</td>
</tr>
<tr>
<td>Connecting an Emulator</td>
</tr>
</tbody>
</table>

*Figure 52*

**Stencyl - http://www.stencyl.com/**

Stencyl’s approach to improve learnability:

1. Easy access to gallery assets

![Stencyl - Super Fun Time]

*Figure 53*
2. Informative help page

![GameBlox](https://gameblox.org/editor/)

**GameBlox** - [https://gameblox.org/editor/](https://gameblox.org/editor/)

GameBlox’s approach to improve learnability:

1. A slide-out tutorial panel on the right

![GameBlox](https://gameblox.org/editor/)
5.2 Format & Structure

5.2.1 Display Format

There are three most effective approaches that similar platforms use to engage new users:

1. Static, step-by-step tutorials with instructions and screenshots
2. Community Q&A forums where questions are answered by staff or other users
3. Video tutorials for sample projects or common operations

Considering the budget and time resource available, the team agreed that adding static TaleBlazer tutorials would be the most affordable, feasible and effective. There are three formats we can use to display TaleBlazer supporting materials:

1. Tutorials in a separate window

![Figure 56: A mockup of tutorials in a separate window](image)

This option will not take up any script editor space and users will have the flexibility of choosing where to place it. However, this architecture does not support interaction
with the editor. Moreover, it will be more like a manual page, which is not ideal for real-time help.

2. Tutorials in a side panel

![Figure 57: A mockup of tutorials in a side panel](image)

Choosing this option allow adding interactions with the editor and will most likely be the most effective format for young users who are not fond of reading separate documentations. The only downside is that this panel will take up some editor space.

3. Inline tooltips

![Figure 58: A mockup of inline tooltips](image)
This option offers instant help and is good for simple instructions. However, it is hard to show sequential instructions including more than 1 element locating at different places on screen.

Based on the results from user interviews and comprehensive discussion with the TaleBlazer team, we decided that adding a tutorial panel on the right (option 2) along with some tooltips (option 3) would be the most effective.

### 5.2.2 Tutorials Structure

Similar to Scratch and GameBlox, each TaleBlazer tutorial consists of a “goal” and a series of “steps”. The goal is what users will achieve after following through the entire tutorial. Each “step” is a small task that typically involves only one or two UI. We try to keep each tutorial fewer than 15 steps to maintain its conciseness and effectiveness.

The main tutorial portal will be a menu that helps users navigate to the tutorial they are interested in. As of this writing, the proposed categories are: Getting Started, Advanced, How-to, Blocks.

- “Getting Started” is a series of tutorials that guide new users through creating their first game.
- “How-to” is a shorter kind of tutorial that help users achieve a specific task (e.g. How to make your agent ‘talk’, How to make a score)
- “In Depth” includes tutorials that explain additional features that are not immediately obvious and require further guidance (e.g. Use TaleBlazer without a data connection).
- “Glossary” explains terminology used in TaleBlazer (e.g. Agent, Region, Action). The “Blocks” section provides description and sample usage of each block in TaleBlazer.
The MVP (minimum viable product) version will only include text instructions and screenshots. Embedding videos can be added in the future to provide better visual support.

5.3 Implementation

I created a PHP Controller which handles the route logics and defines the layout for tutorials. I included a variable called “contained” to denote access via the editor-contained panel. Currently these tutorials are only accessible from the editor. In the future, we can use this variable to adjust the layout of the tutorial pages when tutorials are accessed from the TaleBlazer website directly.

Figure 59: The redesigned tutorial panel expanded
The tutorial slider is implemented as a collapsible panel on the right side of the screen with an iFrame inside that points to the menu page (See Figure 59). The tutorial panel is automatically expanded when users create new games.

A click handler is attached to the close icon in the top left corner, sliding the tutorial container to the right when users choose to close the panel. Once the panel is hidden, the toggle button (which used to be a close icon when the panel was open) turns into a question mark icon as shown in Figure 60. When a user clicks on this small question mark icon or the “Tutorials” button on the navbar, the tutorial panel will slide out. It also remembers the last tutorial in view before being closed, so users do not need to re-navigate to the tutorial they were reading.

![Figure 60: The redesigned tutorial panel collapsed to a question mark button](image)

The menu page contains lists of tutorials by category. Certain list items (e.g. Blocks) have sub-lists that can be expanded on click. To achieve this effect, I added a click handler for these list items that slides up or down depending on whether their sub-lists are visible (See Figure 61).
Each tutorial has its own view (.ctp file) in the same basic structure. First, there is a navbar on top that allows users to navigate between steps. All the steps within the same tutorial are laid out horizontally in a carousel and only one carousel item is visible at a time. Multi-step tutorials start out with a table of contents (TOC) of steps as shown in figure 62. This TOC page is implemented as step 0 in the JavaScript code, providing a preview of the steps included in the selected tutorial. By clicking on a specific step from the TOC page, users will be taken to the selected step with a smooth sliding animation. The navbar shows which step the user is on and allows her to easily move forward and backward between steps. All transitions between steps are implemented using jQuery animation. Each step button (e.g. next, previous, step button in TOC) is attached with different click handlers that take care of three tasks:

1. hide or show buttons (e.g. if users reach the last step, the next button will disappear and the “Go back to Menu” button will appear at the end of the step)
2. calculate how much the carousel should move

Figure 61: Tutorial panel with sub-lists that expand on click
3. change the carousel height depending on the height of the visible step
4. automatically scroll to the top for users when they start a new step
5. update the current step number

![Image](image.png)

Figure 62: Table of Content page for multi-step tutorials (left) and the selected step (right)

The system will detect any tutorials with only one single step and skip the TOC page as shown in Figure 63 (i.e. skip step 0 and show step 1 right away). From implementing this way, I can maintain the same structure for all tutorial templates – single-step or multi-step, and therefore streamline the population process for future tutorials.
To prevent the text being cut off when the content is taller than the panel height, I included a scrollbar and a gradient box at the bottom. The half-transparent gradient box is added to the bottom of the page with a higher z-index to indicate that there is more content below the panel’s bottom edge (See Figure 61 and 62).

5.4 Content Creation

To allow non-technical members to populate more tutorial content in the future, I set up a straightforward system for filling out tutorial templates. All the text, headers and images have relevant CSS classes assigned to them. As long as the content is tagged properly, all the UI interaction including styling will be handled by the JavaScript and CSS code. Since the JavaScript is written generically to handle all situations, all the tutorials share one single JavaScript file. With this scalable structure, I created a stand-alone version that only
contains the relevant JavaScript and CSS files. Anyone can use this stand-alone version to easily populate TaleBlazer tutorials without setting up TaleBlazer’s back-end system.

All that content providers need to do is to properly tag new tutorial content with `<h4>` for headers, `<p>` for paragraph text, `<img>` for images, etc. There are four CSS classes created to help content providers style certain text differently. For example:

- TaleBlazer terminology (e.g. agent, region, action)
- Game specific text (e.g. the “pirate” agent)
- Editor UI elements (clone button, delete button, agents tab, etc.)
- Blocks:

![Figure 64](image)

- Tips with light bulb icon in front of the paragraph:

![Figure 65](image)
• Definitions with book icon in front of the paragraph

Figure 66
Chapter 6
User Testing

To incorporate user-feedback, I utilized an iterative process of prototyping, testing, and refining UI based on user testing results. The initial user test described in chapter 3 helped me gain insights into usability and learnability problems with the prior TaleBlazer editor UI. Throughout the design and implementation process, I prioritized feature requests and UI improvements based on discussion with TaleBlazer’s project manager and developers. Once the basic mockups were created, members of the TaleBlazer team and the STEP lab would review and provide feedback. Upon completion of a digital prototype, I would conduct several users tests to gain feedback from volunteers who are not familiar with TaleBlazer. Results of these small-scale user tests on UI updates are incorporated into the decision making process explained in chapter 4.

As an evaluation of the usability and learnability of the redesigned editor UI, I conducted a larger-scale user test with the new tutorial panel and all other UI changes. This final user test was conducted using my personal computer (MacBook Pro Retina Display, 2.8 GHz Intel Core i7, 16 GB 1600 MHz DDR3) in a Chrome browser. I first gave an overview of TaleBlazer and then asked each user to complete a simple game with at least a map and two agents. Users were instructed to create a game in the editor and then download it onto their phone to play. As a “think aloud protocol”, users were asked to verbalize their thoughts as they interacted with the editor UI. I sat next to the user during the test (observing both the user as well as the screen), recording their actions, pauses and questions without providing any hints.
Ten users who have never used TaleBlazer before were recruited to participate in the user testing. Four out of ten are teenagers between ages 11 to 14, and the others are MIT students with different majors. Important findings from this user testing include:

1. Nine users followed through the tutorials immediately without any verbal cues from me. One of them started to interact with the UI directly without reading the tutorials despite the fact that the tutorial panel is expanded by default. He managed to set up a map and two agents, but eventually referred to the tutorials to learn how to play the game on his mobile device.

2. 80% of users successfully created their first TaleBlazer game without any hints from me. This is a huge improvement in comparison of the 33% success rate from the initial user testing outlined in chapter 3.

3. The implementation of the scrollable script drawers greatly improved the efficiency of finding certain scripts. Users were asked to complete a sequence of five scripts that were randomly selected before user testing. On average, there is a 40% reduction in the time of completing this task by using the newly implemented scrollable drawers. Users appreciated the convenience of being able to scroll through the drawers in the new implementation so that they do not need to correctly remember the corresponding category when looking for a certain script.

4. Two users had trouble setting up the location service on their iOS devices. I resolved this issue for both users by allowing the TaleBlazer mobile app to use location services. This solution involved going into iPhones settings and find the location services hidden under “privacy”. After discussing with the TaleBlazer team leaders, we decided to add a troubleshooting page to the tutorial panel.

5. One user hesitated to switch back to map tab after editing an agent on the Agent tab, expressing his concern that he was not sure if his edits would be saved or wiped out.
As a solution, we could explain in the tutorials that changes between tabs are persisted locally. However, users would have to save the game to update the remote file on the server because TaleBlazer does not save automatically like Google Docs.

6. One user spent a long time adding an image for her agent in the pop-up dialog. She thought the “My Files” tab would include files on the local computer she was using (See Figure 67). She also tried to drag-and-drop her image from the local finder window to the dialog. This is a common feature offered by Google Drive, Dropbox or other file management systems. We could consider redesigning this dialog and supporting drag and drop operations in the future.

![Figure 67: The tabs and upload mechanism in the image pop-up dialog](image)

All users responded positively to my post-user test survey questions. Nine of them agreed that it would have been difficult to complete their first TaleBlazer game and play it on their mobile devices without the tutorial panel. In order to make the tutorials more effective, one user suggested that TaleBlazer add tutorials with sample games to help understand the purpose of TaleBlazer and see potential use cases.
Chapter 7
Future Work

The main purpose of this thesis is to improve the usability and learnability of the TaleBlazer editor. Although significant improvements to the UI have been implemented, there are still a number of features that can be added in the future to further expand the functionality of the TaleBlazer editor.

7.1 Video Tutorials

Many beginners benefit from step-by-step video or animation tutorials where they see exactly what to do to achieve a certain goal. As visual aids are generally more effective than plain word description, adding screenshots, videos and animations for common actions in the development process would be very helpful for many non-technical game designers, especially younger users. The tutorial panel in this thesis only includes text instructions and screenshots, but the architecture is built to easily embed videos in the future. Adding an intuitive video portal to help game designers navigate between tutorials will also be beneficial for novice TaleBlazer users.

7.2 Search Function for Tutorials

The current documentation for TaleBlazer is several PDF files that fail to offer an easy way to search, bookmark or edit. The tutorial panel implemented in this thesis helps new users to get started by following a series of tutorials. However, it does not offer great support for users who have specific questions in mind. In the process of learning a new computer language, many people exploit online Q&A or tech forums to seek help. For new platforms
such as TaleBlazer, which does not currently possess a huge forum, a comprehensive and searchable help page will be very critical for users looking for specific solutions rather than general tutorials. Since TaleBlazer does not currently have as many staffs in comparison to some platforms that offer community forums (e.g. App Inventor and Scratch), adding a search function to TaleBlazer tutorials would be a cost-effective intermediate step before building TaleBlazer Q&A forums.

### 7.3 Interactions between the Editor and Tutorials

The tutorial panel built in this thesis was implemented as a side panel in the editor. With this architecture, future TaleBlazer developers can work on supporting interactions between the tutorials and UI elements in the editor. For example, once a user gets to a certain step in the tutorial with instructions on how to set up actions for an agent, the left panel will automatically expand with the “Add Action” button highlighted. As another example, if the user wants to learn about a certain block, she can click on the block to toggle the tutorial panel with the description and sample usages for that block of interest.

### 7.4 Undo and Redo

Many editor interfaces support undo and redo functionality to allow users to easily recover from mistakes. This thesis prioritized the implementation of script drawers and tutorials over undo functionality based on the user interview results. However, the undo feature will greatly improve the user experience of connecting script blocks. It will allow users to quickly recover from an accidental deletion of multiple scripts and provide a better error recovery mechanism than what TaleBlazer currently supports (i.e. reverting back to an earlier version of the game file).
Chapter 8
Contribution and Conclusion

8.1 Contributions

The contributions of this thesis include:

1. Redesigned UI elements:
   Many UI elements such as buttons, side panels, scrollable containers have been or redesigned to incorporate best practices of modern web design. New components including the navbar and overwrite warnings are added to the editor to provide TaleBlazer game designers a better user experience.

2. Better scripting environment:
   Huge improvements have been made to the script drawers and the toolbar on the scripting page to allow users to seamlessly find, connect, copy, cut, paste, and delete script blocks. These new features resulted in a big increase in efficiency during the final user testing.

3. In-editor tutorials:
   In order to help beginners get started with TaleBlazer, a tutorial panel is added to the editor. During the final user testing, the tutorial panel effectively guided beginners through the process of creating and playing their first game. The scalable architecture also allows other TaleBlazer members to quickly add tutorials and expand its functionality in the future.
8.2 Conclusion

The new version of TaleBlazer’s web editor presented in this thesis was built to allow beginners to easily create their first few simple games, while experienced users can utilize higher functionalities to build sophisticated gameplays. With significant improvements on the editor’s usability and learnability, the goal of this thesis is to help expand TaleBlazer’s user base and provide non-technical game designers an intuitive online editor to learn blocks-based programming and develop educational AR games.
Bibliography

3. MIT Scheller Teacher Education Program. MITAR. http://education.mit.edu/projects/mitar-games
5. MIT Scheller Teacher Education Program. GameBlox. https://gameblox.org/editor/