An Improved TaleBlazer Editor to Encourage and Facilitate Collaboration between Multiple TaleBlazer Game Designers

by

Linda Wang

S.B., Massachusetts Institute of Technology (2016)

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

[Master of Engineering] at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Signature redacted

Department of Computer Science and Engineering
August 16, 2016

Signature redacted

Certified by ............

Eric Klopfer
Director, MIT Scheller Teacher Education Program
Thesis Supervisor

Signature redacted

Accepted by ..................

Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee
An Improved TaleBlazer Editor to Encourage and Facilitate Collaboration between Multiple TaleBlazer Game Designers

by

Linda Wang

Submitted to the Department of Electrical Engineering and Computer Science on August 16, 2016, in partial fulfillment of the requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

TaleBlazer is a platform for creating and playing educational location-based augmented reality games. These games are made by both adults and children game designers using the online TaleBlazer Editor. Oftentimes, building a TaleBlazer game is a group effort, but the previous Editor made collaboration inefficient, difficult, error-prone, and frustrating. This thesis describes a suite of features designed to improve version control for a game and assist designers in sharing their work with others. Together, these features encourage and facilitate collaboration between multiple TaleBlazer designers.

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

I had a wonderful time and learned a lot as a member of the TaleBlazer team. I’d like to thank Professor Eric Klopfer, Lisa Stump, and Judy Perry for giving me this opportunity. Lisa and Judy were quick to give ideas and helpful comments on my work, and Lisa helped me thoroughly test every small detail we could think of.

I’d also like to thank project managers Paul Medlock-Walton and Daniel Wendel and their students. They kept the lab environment lively and willingly offered feedback on my ideas. I’d also like to thank the facilitators of Global Kids and the Missouri Botanical Garden for their feedback from an end user’s point-of-view.

I’d like to thank my friends who support all my crazy endeavors and are endless sources of joy in my life. I’d especially like to thank my partner-in-crime Jin Pan for staying up with me all those late nights, navigating the first steps of post-undergraduate life with me, and always knowing how to make me smile.

Finally, I’d like to thank my parents for teaching me to love and be selfless, work hard and dream big. They have always provided me with the very best of everything my entire life. This thesis is the result of two months of hard work from me and twenty-two years of patience, guidance, and sacrifice from them.
Contents

1 Introduction ................. 11
  1.1 Motivation .................. 11
  1.2 Thesis Summary .............. 13

2 Background .................. 15
  2.1 TaleBlazer Infrastructure ..... 15
  2.2 Tour of the Editor .......... 17

3 Research and Evaluation of Existing Collaboration Tools ....... 21
  3.1 Various Merge Conflict Solutions .......... 21
  3.2 Other Blocks-Based Programming Editors .......... 25
  3.3 Git ........................ 29
  3.4 Google Docs ................ 30
  3.5 Conclusion ................ 32

4 Preliminary Work .............. 37
  4.1 User Interviews .............. 37
  4.2 Finalized Suite of Collaboration Tools .......... 45

5 Tools for Version Control ....... 49
  5.1 Saving Experience .......... 49
  5.2 Conflict Handling .......... 52
  5.3 Undo/Redo ................ 56
  5.4 View Changes ............... 65
5.5 Revision History ................................................. 70

6 Tools for Sharing ................................................. 77
  6.1 Remix .......................................................... 77
  6.2 Look Inside .................................................... 84
  6.3 Copy/Paste ..................................................... 86

7 Future Work ....................................................... 113
  7.1 Improving the Existing Suite ............................... 113
  7.2 Expanding the Suite ......................................... 115

8 Contributions and Conclusion ...................................... 119
  8.1 Contributions .................................................. 119
  8.2 Conclusion .................................................... 120

A Preliminary User Interviews ...................................... 121

B Discussion of Remix, Save As, and Copy/Paste ..................... 125

C Discussion of Undo/Redo, View Changes, and Revision History 131
Selected List of Figures

2-1  The previous TaleBlazer Editor, annotated .......................... 17
5-5  The updated conflict dialog ............................................ 55
5-11 View Changes, which displays a list of changes made in the game 66
5-13 The Editor’s previous and updated navigation bars .................. 68
5-18 The updated interface for restoring a previous version of a game . 74
6-8  The Remix and Look Inside landing page .............................. 85
6-12 Some possible outcomes when pasting a script ...................... 91
6-23 Copy/Paste UI mockup 1 .............................................. 106
6-24 Copy/Paste UI mockup 2 .............................................. 107
6-25 Copy/Paste UI mockup 3 .............................................. 108
6-26 The finalized Copy/Paste interface ................................. 110
Chapter 1

Introduction

Augmented reality is the increasingly popular concept of enhancing a real-world environment with computer-generated data, creating an interactive experience for the user. The TaleBlazer platform uses augmented reality to educate children through location-based games. After downloading a game through the TaleBlazer mobile app on a GPS-enabled smartphone, players can encounter virtual characters, objects, and environments while they move around the real world.

TaleBlazer's combination of education and gameplay allows zoos, museums, summer camps, and other programs to enhance students' experiences and help them engage with their environment while learning material. Some of its biggest partnerships include the Columbus Zoo and Aquarium and the Missouri Botanical Garden.

TaleBlazer games are not only designed for children, but are often designed by children as well. Programs such as Global Kids and the Missouri Botanical Garden encourage students to design their own games and build them using TaleBlazer's online Editor. Youth designers are usually of middle to high-school age attending week-long summer programs or weekly afterschool programs.

1.1 Motivation

Shortly after I joined the TaleBlazer team, I had the pleasure of brainstorming game ideas the US China Scitech Education Promotion Association (SciTech), a group of
educators from China. Our goal was to design a series of three-hour games to be played by hundreds of students during field trips to Beijing’s DongFeng park. With an ambitious schedule of building these games from scratch in just three months, the group would have benefited from an efficient way to work together to meet the deadline.

Like the SciTech games, many TaleBlazer games are created through the group effort of several people, by both adult and youth programmers. However, the previous TaleBlazer platform was neither designed for nor encouraged the model of multiple people working on a single game. The online game Editor restricts a game’s edit access to only a single account, and while it allows a game to be edited simultaneously on two computers, it provided no way of synchronizing those changes.

From my experience working with the SciTech team and after interviewing program facilitators who have mentored children building TaleBlazer games, it was clear that both adult and children designers would benefit from features that assist them in working with others — something that the previous TaleBlazer Editor made error-prone, time-consuming, inconvenient, and even exasperating at times. The goal of my thesis work is to improve the Editor to both encourage and facilitate multiple game designers working together on a single TaleBlazer game. To achieve this goal, I focus on improving the Editor in two main areas: version control and sharing.

When multiple people collaborate on a single project, the ability to view changes and restore previous versions is not only useful but necessary. This is the idea behind version control. The previous Editor’s version control tools were lacking in both functionality and presentation. One common problem resulted from two designers opening the same game on their computers and making changes simultaneously. The Editor, unable to reconcile the differences, would end up overwriting one designer’s changes with the other’s. As a result, TaleBlazer designers who wished to collaborate on a single game needed, work at different times, communicate clearly, and save often; otherwise, a lot of work had the potential to be lost with no way of being restored. Not only was this difficult and inefficient, losing work was especially upsetting for children and sometimes led to tension within a group. In addition, a single game
designer who accidentally edited his game in two different browser windows (e.g. on a home computer and a work computer) faced overwriting his own changes as well.

Once a game — or part of a game — is completed, the game designer may wish to reuse his work in another game. Other game designers would benefit from this as well. This is the idea behind sharing. Some examples of this are:

- A player enjoys a TaleBlazer game that she downloaded and wants to create something similar.

- Children working with TaleBlazer for the first time want to experiment with working parts of an existing game.

Currently, game designers are able to view the source code of certain games, but there is no way to reuse a specific part of that game. They would need to either copy the entire game and delete the unwanted sections or recreate the desired pieces from scratch.

I took on the challenge of mitigating these existing issues by designing and implementing a suite of tools to provide TaleBlazer designers with

1. more efficient and user-friendly methods of version control, and

2. features to aid sharing their work, in whole or in part, with others

I discussed the functionalities and interfaces of these tools with several users of differing experiences with TaleBlazer and game design. I carefully analyzed and incorporated the feedback I received, and the resulting Editor at the completion of my thesis provides more intuitive, efficient, and elegant methods of collaboration.

1.2 Thesis Summary

Chapter 2 outlines the main areas of the TaleBlazer infrastructure and gives a detailed overview of the online Editor.
Chapter 3 discusses various modern-day products and the tools they provide to help their users collaborate with one another.

Chapter 4 summarizes interviews with three TaleBlazer users, explains the problems of the previous Editor, analyzes suggested solutions, and finalizes the suite of tools completed in this thesis.

Chapters 5 and 6 detail the tools for version control and sharing, respectively. Each tool solves a different problem and came with its own challenges. These chapters explore the design decisions that were made in making each tool production-ready.

Chapter 7 suggests ideas to further improve the Editor’s ability to aid collaboration.

Chapter 8 concludes with a summary of the work completed in this thesis and its impact.
Chapter 2

Background

This section reviews the main parts of the TaleBlazer platform, with a focus on the Editor. While most of my thesis work focuses on improvements to the Editor, it involves parts of the Server as well.

2.1 TaleBlazer Infrastructure

The TaleBlazer platform is comprised of four main parts: Editor, Server, Mobile, and Analytics.

2.1.1 Editor

TaleBlazer game designers create games using the TaleBlazer online Editor, a web application that features a blocks-based programming environment. Designers can define their game’s location, introduce agents — virtual characters and items that appear in certain places and under certain game states — and keep track of player and game stats. In addition, designers drag and arrange customizable blocks of logic to execute under different conditions and possibly change the game state. These blocks can be nested (e.g. conditionals and loops) and attached (e.g. several steps of logic to be executed serially) to create a script. A single agent can have several scripts that execute under different conditions. With scripts, developers can create sophisticated
games with intricate game mechanics. A more detailed tour of the Editor is provided in Section 2 of this chapter.

2.1.2 Server

The Editor is located within TaleBlazer's website. The website provides resources for working with TaleBlazer, including documentation, tutorials, and information about TaleBlazer and partner organizations. Designers also use the website to access their games and games created by others. As part of my thesis, I added new pages in the Server to view a game’s Revision History (Chapter 5 Section 5), and to Remix (Chapter 6 Section 1) and Look Inside (Chapter 6 Section 2) a game.

2.1.3 Mobile

TaleBlazer apps are deployed to a smartphone, where players can download specific games by either:

- downloading a game of their own after signing in
- inputting the game code of another user’s game, or
- standing in close proximity to a site that has partnered with TaleBlazer to offer games to the general public

Players build their own experience with the game based on their location and in-game decisions. TaleBlazer currently supports both Android and iOS.

2.1.4 Analytics

TaleBlazer makes use of its own analytics website to track and analyze gameplay data, such as the total number of games played or how many players talked to a specific agent within the game.
2.2 Tour of the Editor

The Editor was previously designed and optimized for editing games developed by a single creator. My thesis work adds new features and improves upon already existing ones to build a strong suite of tools in order to aid designers in version control and sharing games.

Figure 2-1 shows an annotated screenshot of the previous TaleBlazer Editor. The rest of this section explores the most important parts of the Editor from the top of the screen to the bottom. Each part is used for either navigating to different parts of the Editor or for editing the game.

Figure 2-1: A screenshot of the TaleBlazer Editor in its previous state, open to the Agents tab and focused on the Apple agent. The script editor shows two scripts.
2.2.1 Navigation Bar

Purpose
Navigation

Details
This is the dark gray bar located at the top of the Editor. It previously contained seven buttons to help designers navigate the TaleBlazer site and to assist them in building the game. This bar includes the Save button, discussed in Chapter 5 Section 1. The navigation bar’s interface was updated as part of my thesis work to include buttons to allow designers to undo and redo (Chapter 5 Section 3) and view their changes (Chapter 5 Section 4).

2.2.2 Tabs

Purpose
Navigation

Details
The Editor has six tabs, each of which populates the Editor with a different view for designers to edit different parts of their game. Designers spend the bulk of their time and do the majority of their work in the Agents, Map, and Player tabs.

Map Tab — Designers use this to create regions that determine where the game is located. The Map tab includes an embedded Google Maps widget with a satellite view of the region. Here, designers can view where agents in the region are located, and click and drag on a specific agent to relocate it. Designers can also define a region’s boundaries by clicking and dragging markers on the map, specify a region to be indoors, and upload a custom image on top of the Maps widget.

Agents Tab — Designers use this to create agents, or virtual characters and items, that appear in their game. The Editor is open to this tab in Figure 2-1. The Agents Tab consists of four main areas: the ribbon, the dashboard, and the script area. Each
of these areas are explored in more detail later in this section.

Player Tab — The player can sometimes assume different roles, defined in this tab. Like the Agents Tab, this tab is made up of a ribbon, dashboard, and script area.

World Tab, Settings Tab — Designers can configure settings pertinent to the game world and to gameplay in these tabs, respectively.

Beacons Tab — The designer uses this tab to configure Bluetooth beacons for indoor gameplay.

### 2.2.3 Entity Ribbon

**Purpose**

Navigation

**Details**

This ribbon appears in the Map, Agents, Player, and Beacons tabs where it displays all of the regions, agents, roles, and beacons in the game respectively. Located above each ribbon is a button to create a new entity based on which tab is currently open.

### 2.2.4 Entity Dashboard

**Purpose**

Editing

**Details**

The dashboard is present in the Agents, Player, and World tabs. Here, designers can edit the entity’s name, image, description, actions, and traits. For agents, the dashboard is also used to set location and bump settings which define where and how players encounter the agent.
2.2.5 Script Drawer and Script Editor

Purpose
Editing

Details
The Agents, Player, and World tab contain a blocks-based programming environment known as the script editor. Designers select blocks from the script drawer and drag them onto the script page to build scripts. The script editor in Figure 2-1 is made of two scripts, the bottom of which is made up of several nested and connected blocks. The script editor contains all of the scripts for the selected entity and is implemented via a shared library called ScriptBlocks.
Chapter 3

Research and Evaluation of
Existing Collaboration Tools

Collaboration is an important skill often required in schools, workplaces, and other areas of life. When working together, two team members might make changes that conflict with each other. This chapter first documents a few approaches to mitigate this conflict taken by different products, including TaleBlazer’s previous Editor. This chapter then reviews several blocks-based programming platforms, which face similar challenges due to the unique nature of programming using blocks. Sections 3 and 4 evaluate selected features offered by Git and Google Docs, two widely-used products that facilitate collaboration. Finally, this chapter concludes with how the research in this section relates to my thesis work.

3.1 Various Merge Conflict Solutions

One of the biggest problems with the previous Editor was the way it handled conflicting edits for a game. A TaleBlazer game would run into problems if the following sequence of steps happened:

1. Designer 1 opens the game on his computer. Designer 2 opens the same game on her computer. At this point, the game looks the same on both computers.
2. Designer 1 and Designer 2 make their changes to the game independently of each other. At this stage, neither one knows that the other is working on the game at the same time.

3. Designer 1 saves his game. After a period of time, Designer 2 saves her game as well. There is now a conflict.

The following modifications to the above scenario would also result in a conflict:

- More than two designers are working on the game
- Designer 1 and Designer 2 are a single person who has opened the same game multiple times, e.g. in different tabs, browser windows, or computers

In fact, most conflicts result from a single designer accidentally opening the game twice and making edits in both instances of the game. However, for the purposes of clarity and brevity, the rest of this paper assumes the original outlined situation when discussing conflicting edits within a game: two designers working on separate computers with Designer 1 saving first.

![TaleBlazer](image)

Figure 3-1: With the previous Editor, Designer 2 would receive this pop up within 5 minutes of the game being saved elsewhere.

Under its previous design, the TaleBlazer Editor checked for a more up-to-date version of the game every five minutes. Designer 1 would create a more up-to-date version of the game when he saved. Once this new version was detected, Designer 2 would receive the alert shown in Figure 3-1 asking her either to keep her version of the game or to refresh to show Designer 1's work. This would result in one of the
designers losing his/her work. While not explicitly stated as an option, Designer 2 could also choose to close the dialog and avoid both choices, however, this would only result in the dialog appearing again after 5 minutes.

With this behavior, there could also be up to a five-minute window between the time when Designer 1 saves and the time when Designer 2 is alerted. If Designer 2 saves her work between these two points, she would silently overwrite Designer 1’s work without alerting either designer. Designer 1 would then either receive the same alert up to five minutes later, or worse, if he had exited the game after saving, would not find out that his work had been overwritten.

I paid particular attention to how several products handled merge conflicts. The following products present the conflict and ask the user to handle it manually.

### 3.1.1 Overleaf

Overleaf is an online real-time collaborative editor for LaTeX documents that allows several users to edit a single LaTeX document at the same time; however, if one user’s computer goes offline briefly and reconnects to the server after the document has been changed, she will see the message in Figure 3-2. The document locks and asks the user to refresh the page, and she is unable to make any changes. Although forcing the user to lose her work is not very desirable, this prevents the creation of conflicting versions of the document.

![Overleaf Editor](image)

**Figure 3-2:** Users of Overleaf receive this message when the server detects that their version of the document is outdated. To the left is the LaTeX editor, which is locked and does not allow any further changes.
3.1.2 Jupyter Notebook

Jupyter Notebook is a web application that allows users to collaborate on documents containing code and visualizations. Upon conflict, users are given the same three choices as the original TaleBlazer Editor: losing their work, overwriting someone else's changes, or doing nothing. Jupyter provides all three options as buttons, which makes it easier for users to choose one of the options (see Figure 3-3). In addition, these buttons are color-coded, which not only separates the options but also adds a default desirability for each option: for example, it generally better for a user to overwrite her own work because she has a sense of what she has changed, has the ability to reproduce the work, and does not risk angering another user.

![Figure 3-3: Jupyter Notebook provides the user's three choices as color-coded buttons.](image)

3.1.3 Atlassian Confluence

Atlassian is a company that builds tools for teams that has built several widely-used products including BitBucket, SourceTree, and HipChat. Confluence is one of its products that allows a team to organize and collaboratively edit documents. When Confluence encounters a scenario like the one described at the beginning of this section, it first tries to automatically reconcile edits, but if it fails to do so, it asks Designer 2 to manually fix the conflicts. Confluence helpfully displays the differences between the user's version of the document and the most up-to-date version of the document at the top of the page. Below the error message, the user has access to the document's editor and can fix the conflicts while seeing what they were and where
they occurred. At the bottom of the page, she also has the option to either overwrite the existing document with the changes in the editor or to cancel saving her changes.

Figure 3-4: Users of Atlassian’s Confluence receive this message upon conflict.

3.2 Other Blocks-Based Programming Editors

Blocks-based programming environments are becoming increasingly popular platforms for beginner programmers — especially children — to create coding projects and learn essential programming skills. With blocks-based programming, designers drag, drop, and rearrange blocks of code instead of typing each line of code separately. These blocks are often color-coded based on type and shaped based on legal conjunctions. This nearly eliminates common programming errors due to syntax and typos, lowering the learning curve. However, blocks-based programming faces its own challenges in user experience and design due to its drag-and-drop nature. I investi-
gated several other blocks-based programming platforms, focusing on their solutions to the challenges I faced as well.

### 3.2.1 Scratch

Scratch is a well-known blocks-based programming language developed in the MIT Media Lab's Lifelong Kindergarten Group (LLK). It is designed to teach children, mostly of ages 8-16, to “think creatively, reason systematically, and work collaboratively” [1]. To encourage this last goal, Scratch allows **remixing** — one Scratch game designer can create a copy of someone else’s project and then add his own edits to make the project his own. With this feature, users can “learn from, experiment with, and add on to the work of other users” [2].

To aid users in remixing projects, Scratch offers a **backpack**. Scratch users can open a project and drag and drop sprites (Scratch’s equivalent of a TaleBlazer agent), scripts, costumes (visual additions to a sprite), and sounds from the project into their backpacks. Figure 3-5 shows a backpack with one of each. Users can then open a new project and drag an object from their backpacks into this new project [3].

![Backpack](image)

**Figure 3-5**: An example of a user's Scratch backpack which contains a script, costume, sprite, and sound in that order.

In order for users to add objects into their backpack, they must be able to view the source project in the first place. To do this, users can **see inside** a Scratch project, which opens a project without remixing it.

Finally, Scratch users can download sprites, costumes, and sounds and save them as files on their computer. They can upload these files into another Scratch project instead of using the backpack, if they so choose, as shown in Figure 3-6.
3.2.2 GameBlox and LocalStorage

GameBlox, like TaleBlazer, is a part of MIT’s Scheller Teacher Education Program (STEP Lab). Targeted toward middle schoolers and older children, GameBlox provides a blocks-based programming interface to help designers create and play games on the web and mobile devices. Unlike the TaleBlazer Editor which uses ScriptBlocks, the GameBlox Editor is built on top of Google’s Blockly, an open-source blocks-based programming editor library. As a result, GameBlox’s Editor offers different features than TaleBlazer’s does. One noticeable feature is copy/pasting blocks across games — in the GameBlox Editor, users can right-click on a stack of blocks in one game to copy them, and right-click in another game to paste them.

Copy/paste in GameBlox is implemented using local storage offered by the Web Storage API, which allows developers to store, modify, and delete data for a particular domain [4]. Data is stored in the form of key-value pairs, both of which are strings. There are two types of web storage: session storage data is specific to the browser tab, and it is deleted when the user exits the tab. In addition, “opening a page in a
new tab or window will cause a new session to be initiated”, so a second tab opened in same domain can not access the variables defined in the first [5]. Data stored in local storage, on the other hand, has no set expiration [6]. Thus, the latter is useful for transferring data between two open sessions of a particular domain — perfect for copying and pasting information across games.

The GameBlox Editor stores data in local storage when copying a block, and uses this data to rebuild the block upon pasting. One downside to this method is that it is increasingly computationally expensive to recreate larger and more complicated blocks. While the small stack of five blocks in Figure 3-7 consistently takes less than 100 ms to paste, a stack of about 200 nested and connected GameBlox blocks takes consistently between 5-7 seconds. During this time, the browser freezes and the user can not do anything else. A human’s perceptual processor has a cycle time of about 100 ms [7], so a user will likely not notice the delay if blocks paste within this time frame. Despite the slight disadvantage of having to recreate the block, storing data in local storage provides a simple and effective way of sharing data between two games.

3.2.3 StarLogo Nova and ScriptBlocks

The TaleBlazer Editor is built on ScriptBlocks, a blocks-based programming library it shares with another software platform in the STEP Lab called StarLogo. StarLogo focuses on helping young users create games and simulations to further their understanding of concepts in math and science, specializing in multi-agent models and complex systems. Shortly before my the start of my thesis, ScriptBlocks launched a new right-click menu similar to the one found in GameBlox, allowing users to duplicate, cut, copy, and paste blocks within the same game.

![Figure 3-8: The ScriptBlocks right-click menu to copy and paste blocks.](image)

28
3.3 Git

Git is a popular version control system for software development, and it offers several useful features to aid users in keeping track of different versions of their projects. While Git is used mainly for version control, its features allow users to effectively collaborate and work on a single project simultaneously. With a combination of the features described in this section, users can easily see the changes made by another member, the state of the project before and after those changes were made, and combine those changes with their own. The ability to accomplish all three of these actions is invaluable when multiple people work on the same project.

3.3.1 Branches

As described in Git’s online documentation, branching allows a user to “diverge from the main line of development and continue to do work without messing with that main line” [8]. Branching is typically used when a user wants to make edits to the project without directly affecting the project until he is ready. Users can create a branch from the main line, work on their changes, and finally merge their work into the main line once it is complete. Essentially, two users can start off with the same version of a project and edit that version without affecting one another.

3.3.2 Commits and Logs

In Git, users can commit their changes, which essentially records the changes and provides a checkpoint for their work, along with a commit message, a short, user-written description of the changes. At any time, users can view a log of all of the commits and messages and revert to a specific commit. This is useful if the user adds something he no longer wants, or he accidentally breaks the project.
3.3.3 Diff

With Git, users can use the `diff` command to view changes between different versions of their project. This is useful for finding the source of error if a new change does not go as expected, or for simply reviewing changes made if a user has forgotten what he has worked on. The default differing behavior allows a user to see the changes he has made since his last commit.

3.3.4 Merge

After a user makes changes in a branch, he can use Git to merge back into the main branch. This applies his changes to the state of the project that he branched off of, and the end result is designed to be equivalent to if the user had worked directly on the main branch. Git tries to automatically resolve any conflict between versions of the project, but sometimes it requires the user to manually fix differences.

3.4 Google Docs

Team members looking to collaborate on writing a paper can do so easily with one of several modern-day products. Google Docs is one of the most popular online real-time collaborating environments and offers several useful features, allowing users to simultaneously edit a single document without conflicts. The TaleBlazer Editor’s end goal strives for an experience similar to Google Docs: multiple users can sign into their own accounts and simultaneously edit a single game, and everyone’s changes are accepted and visible in real-time. This end goal is infeasible to complete over the course of a single thesis, but the suite of tools I have put together in my thesis bring the Editor a couple steps closer to this final goal.

3.4.1 Autosave

Google Docs automatically and continuously saves changes as a user is editing. This prevents the case where a user produces a large amount of work, accidentally closes
the browser or navigates away from the document, and loses everything he has done because he didn’t save his work. TaleBlazer users mentioned that this is a recurring issue for young TaleBlazer developers, especially as more products provide an autosave feature, children become less likely to remember to proactively save their work themselves.

3.4.2 Real-Time Merging

Google Docs automatically merges changes made by several users through the use of operational transforms, a technology originally developed for controlling concurrent actions on a single text-based document. Operations are actions that users take to modify the document (e.g. insertions or deletions), and each of these operations are transformed against previous operations in order to keep the document consistent (e.g. if one user deletes a word in front of the document right before the second user inserts text, the index of the second user’s operation must update to reflect the missing letters of the now-deleted word) [9].

Unlike Git, Google Docs does not have the concept of branching, and users make changes directly on a shared copy of a text document. These changes are merged and displayed immediately as they are made, and users are neither required to nor allowed to handle conflicting actions manually.

3.4.3 Revision History

While it is mainly used for collaborating in real-time, Google Docs also offers version control for a document through its Revision History feature, allowing users to view and revert to previous drafts of the document. The right portion of the revision page displays a list of draft timestamps along with a list of people who contributed to the draft. Clicking a timestamp refreshes the page to show the corresponding draft of the document on the left side of the page. Additions to the text are highlighted, while deletions have a strike-through. Each of the edits is displayed in a color unique to the person who made the edit. Figure 3-9 shows an example of a document’s revision
history page, with edits made by Lisa and me at the same time. My changes are highlighted in green, and Lisa’s changes are in purple.

Figure 3-9: The Google Docs Revision History page shows the document’s drafts on the right. Selecting one displays the draft on the left and reveals an option to revert to that draft.

By default, revisions that were made within a time period are grouped together into a single draft, though users can choose to show more detailed revisions to shorten the time frame of grouped changes. After selecting a previous version, the user is given the option to restore that version. This will revert the document to the selected version and refresh the page to show the updated document [10].

3.5 Conclusion

Researching existing products and their solutions was invaluable in brainstorming and designing features for the TaleBlazer Editor. This section summarizes how I used the previously described research in my thesis.

3.5.1 Overleaf, Jupyter, and Confluence

This chapter reviewed approaches taken by three different products.
Overleaf’s solution of locking the document and forcing the user to refresh is simple and effective, but it is undesirable for the user to always lose their work; I felt that there were better solutions for the TaleBlazer Editor.

Jupyter’s presentation of the conflict was color-coded and provided buttons for each option. I used both of these ideas in my work with handling conflict in the Editor (Chapter 5 Section 2.2).

Confluence displays the differences between the conflicting versions of the project, which influenced my ideas of diffing versions of a game. While I proposed this idea to users, I did not end up implementing it; instead it is included in my suggestions for future work (Chapter 7 Section 2).

3.5.2 Blocks-based Programming Editors

The blocks-based programming platforms reviewed in this chapter offered several interesting features.

Like Scratch, TaleBlazer offers a Remix feature, which I revamped and included in my suite of collaboration tools (Chapter 6 Section 1). Scratch’s See Inside feature also greatly influenced the idea and design of the Look Inside feature I added to the Editor (Chapter 6 Section 2).

GameBlox implements copy and paste across games through local storage. I also use local storage implement copy and pasting across TaleBlazer games not only for scripts but for agents and other TaleBlazer entities as well (Chapter 6 Section 3).

Before the start of my thesis work, ScriptBlocks offered a right-click menu to copy and paste blocks, however, users were limited to pasting only in the same game from which they copied the blocks. As part of my thesis, I worked directly in the ScriptBlocks library to update the copy and pasting functions to work across different games (Chapter 6 Section 3).
3.5.3 Git

Git influenced several of the tools that I devised for improving version control in TaleBlazer games.

The Save As feature (Chapter 5 Section 2) is based off of the idea of branching, which allows two users to start off with the same game and make edits that do not conflict with each other.

Users can write a commit message when they merge their branch, which inspired my idea of allowing designers to write comments when saving the game. While not included in my thesis, it is a good feature to have in the future.

Git's diffing feature allows users to see what lines of text they have edited since their last commit, and similarly TaleBlazer designers now benefit from a View Changes feature (Chapter 5 Section 4) which allows them to view a list of changes they have made to the game since opening it.

Finally, Git attempts to automatically merge changes, which is incredibly useful for combining work done in several places, both for multiple users working on a project at the same time and for a single person working in different workspaces. However, the merge process is complicated in both implementation and behavior — for example, in the Editor, it is unintuitive what should happen if one designer performs a lot of work on an agent that another designer deletes. While a fully working merge feature for TaleBlazer was out of the scope of my thesis work, the suite of tools that I worked on — such as a better saving experience (Chapter 5 Section 1), improved conflict handling (Chapter 5 Section 2), and the new View Changes feature (Chapter 5 Section 4) — will act as stepping stones for a merge feature in the future.

3.5.4 Google Docs

The TaleBlazer Editor's end-goal is a Google Docs style of collaboration.

Google Docs's autosave feature is incredibly useful and takes the pressure of saving off of the designer. However, it is a double-edged sword, as the Editor has no way of knowing which changes are intended to be kept. For example, without
autosave, a designer might experiment with a new feature and simply not save the
game if it does not work as intended; with autosave, all of these changes will be
automatically saved. In order to provide designers with a simple way to remove these
experimental changes, TaleBlazer needs to allow designers to undo and redo, which I
implemented as part of my thesis work (Chapter 5 Section 3).

Like Git’s merge feature, Google Doc’s real-time merging and updating is
strongly desired and incredibly useful, but difficult to design and implement well. A
merge that works only part of the time or combines only some of the changes is unre-
liable and undesired. Furthermore, Google Docs is designed for users to collaborate
on a single text-based document, whereas the TaleBlazer Editor allows designers to
make both text-based and visual-based changes. Designers can also be in different
tabs and make changes in one tab that affect another (e.g. after deleting a region, all
agents that were previously located in the deleted region will be moved to another
region). However, once these difficulties are overcome, a fully functional Google Docs
version of TaleBlazer that automatically updates and merges the Editor would create
an efficient work environment for a team of people working on a single game, and is
an end-goal to strive toward.

While the Google Docs’ first two features were outside the scope of my thesis
work, TaleBlazer offers a Restore feature that, like revision history, allows users to
view and revert to previous versions of a game. While interviewing TaleBlazer users
to discuss what features they would find useful, one user asked for a Restore feature,
not knowing that one already existed. Based on the Google Docs revision history
interface, I built upon the existing Restore feature to make viewing a game’s revision
history and restoring a specific version more accessible and user-friendly (Chapter 5
Section 5).
Chapter 4

Preliminary Work

After researching solutions offered by existing technologies, I came up with several possible features that I felt would be useful additions to the TaleBlazer Editor. I conducted interviews with three TaleBlazer users who have had experience facilitating groups of children working together on a single TaleBlazer Game. We discussed what the previous TaleBlazer Editor did well to allow collaboration, what could be improved, and how they might use the suggested new features (if at all). Based on the results of these interviews, I determined the suite of tools that I would create and improve to make it easier for TaleBlazer designers to work together. This chapter details the interview protocol, each of the suggested features with a summary of the feedback it received, and the finalized suite of features I completed during my thesis work based on the users’ responses.

4.1 User Interviews

4.1.1 User background

Two of the users interviewed are former facilitators of the Global Kids program. Global Kids organizes summer programs centered around the creation of TaleBlazer games where up to ten children work together on a single game. To avoid running into conflicts, the program has come up with a few solutions, but each solution generates
its own problems:

- **Solution**: Children work on a single master account one at a time.
  
  **New Problem**: This is inefficient and it is difficult to find meaningful tasks for children who aren’t working in the master at the time.

- **Solution**: Children work on different aspects of the game, e.g. creating the map or writing the storyline.
  
  **New Problem**: Not everyone gets to experience actually programming the game.

- **Solution**: Children work in different games at the same time. Facilitators then manually merge everyone’s work into a single master game.
  
  **New Problem**: This is not only time-consuming and tedious but also prone to error, as it is easy to miss small details like a checked checkbox.

Despite its disadvantages, Global Kids most commonly uses the last solution.

The third user facilitates youth creation of TaleBlazer games at the Missouri Botanical Garden. In this program, students work in smaller groups of two or three, so merge conflicts are not as much of an issue. However, children sometimes accidentally opened up two tabs and make edits in both, causing conflicts. The facilitator has also worked extensively with adults designers creating TaleBlazer games for professional use.

### 4.1.2 Interview Protocol

Each interview followed a detailed interview protocol which went through several iterations of revisions with Lisa Stump, TaleBlazer’s lead software developer, and Judy Perry, TaleBlazer’s project manager. The protocol began with an introductory phase where I stated my main goal of making collaboration easier for TaleBlazer designers. I asked the interview subject to describe their experience using the TaleBlazer Editor, specifically focusing on times when multiple children were working on a single game. I then solicited their feedback for my suggested ideas, and we discussed the
advantages, disadvantages, and use cases for each one. Finally, I concluded by asking the user to rank the features and asked for any ideas of his/her own.

The interview questions were designed to be informative enough for the users to understand the feature, and general enough for the users to come up with their own use cases for the feature. To encourage honest feedback, the questions were worded to limit the opportunity for induced bias. For example, we replaced a sentence like “I brainstormed some ideas that I would like to introduce to you”, which is more personal and is more likely to encourage positive responses, with “I would like to introduce some ideas to you”, which is more indirect and is more likely to produce honest feedback, whether positive or negative.

A summarized version of the general interview protocol can be found in Appendix A. It has been condensed to be more concise than the actual interview, which went into more detail, especially if the user asked clarifying questions. The questions asked during the interview were also slightly modified depending on the user’s prior experiences and his/her responses to earlier questions.

The interviews were conducted over a remote video conferencing software and lasted about one hour each.

4.1.3 Summary

Having previously only observed the adult SciTech team making games with TaleBlazer, I did not have any experience with children using the TaleBlazer Editor. Speaking with TaleBlazer users who have worked with children developers was very useful in imagining the other user class using the Editor.

In general, users felt that the previous TaleBlazer Editor’s biggest problem was allowing a student to overwrite another student’s work. This was very upsetting, especially for children, and sometimes led to tension. It is very important to prevent this as much as possible and mitigate the consequences if it happens. In addition, the Global Kids program (and likely other programs working with large groups of children) would benefit from a way to efficiently recreate parts of one game within another game.
The rest of this section details the list of features I presented to the user. For each one, I briefly describe the feature and the user feedback I received. The features are listed in decreasing order of positive response, from most to least valuable.

Each of the features assumes the following the setup:

1. John and Sally are working on a Halloween-themed TaleBlazer game together. They both open the game around the same time, on their separate computers.

2. John creates a Witch agent, while Sally creates a Black Cat agent.

3. John saves his work.

4. With the previous TaleBlazer Editor, Sally is eventually shown the popup described in Chapter 3 Section 4.1. She has to decide whether to keep her Black Cat or refresh the page (showing the Witch, but she will not know this until she refreshes the page).

Some of the following features are designed to help Sally make a choice or to mitigate the conflict as much as possible. Other features are designed to assist Sally with creating the game, with John out of the picture.

4.1.3.1 Copy/Paste Agent

Description
A designer can copy and paste an agent across game files.

Examples

- Sally can open the Halloween game again in a new tab, displaying the most recently saved version of the game with John’s Witch. Sally can copy her Black Cat from the original tab and paste it into the second tab, combining her changes with John’s. She can save in the second tab and then close the original tab, discarding the changes there.
• Sally decides to create a Thanksgiving game and wants to reuse the Black Cat — she can copy/paste it from the Halloween game without having to recreate it entirely.

Feedback

Users really liked this idea, and it turns out that this was a feature that had already been previously requested by other TaleBlazer users. They were very quick to come up with even more use cases for this feature:

• Facilitators can create template games that have a suite of different agents with different functions. Students can then pick and choose which agents they want in their own game and experiment with different types of agents without having to recreate everything themselves.

• As before, students would create their own agents in separate games, but now facilitators can directly copy a student’s work into the master copy instead of tediously recreating each one. This would greatly increase efficiency and reduce errors, as it is easy to miss checkboxes and other small details in the student’s original copy.

• The Editor restricts edit access for a game to a single user account, but students like having their own account to differentiate their work from their peers’. With copy and paste, students would be able to work in their own accounts and combine their work into a separate, shared account.

4.1.3.2 Merge Changes

Description

If a game is edited by two designers at the same time, this automatically merges the changes where possible and asks the designer to do so manually where necessary.

Examples

• Sally sees John’s Witch appear as she is working on her Black Cat.
Sally sees an error message explaining that the Editor was unable to merge the changes. She would see that in her version of the game, she has a Black Cat, whereas in the saved version of the game, there is a Witch where her Black Cat is. She is asked to fix the errors and save, or refresh the page and overwrite her changes.

**Feedback**

Users felt that being able to keep both students work without having to go through any extra steps would be the ideal scenario. They also felt that this would encourage collaboration, which is something the programs want children to learn and practice. One drawback is that students would be forced to accept another student’s changes. Otherwise, the feedback was overwhelmingly positive, though users stressed that this feature must be done well in order to be useful.

### 4.1.3.3 Save As

**Description**

If a game is edited by two designers at the same time, this saves both copies of the game as separate games.

**Example**

- Upon seeing the conflict message, Sally can choose the option to save her current state of the game as a different version, essentially branching off the main version. There is now two different versions of the game — one with only the Black Cat, and one with only the Witch.

**Feedback**

Users felt that this feature would be useful in the case of merge conflicts. Both children would be able to keep their work, and they can discuss their thoughts in order to reach a consensus. Users thought this was a better solution than the current method of overwriting one person’s work, but also felt that it would be less useful.
if a merging algorithm already exists. One user also mentioned that this would be helpful if two students wanted to take an existing game down separate paths.

4.1.3.4 Display Changes

Description
Upon receiving the conflict message, the designer sees a list of changes that she has made since she last saved the game file.

Example

- When Sally receives the conflict message, she can see that she added the Black Cat. She now knows that refreshing the page will delete her Black Cat.

Feedback
Users felt that this feature would be useful if someone had made significant changes without saving. They also felt that it would be helpful to see the conflicting changes as well.

4.1.3.5 Undo/Redo changes

Description
This allows the designer to undo her last action within a session as well as redo an undone action.

Example

- Sally can delete her Black Cat agent. She can then change her mind and undo, bringing her Black Cat back into the game.

- Sally can also redo the deletion, removing the Black Cat once more.

Feedback
Users felt that this is a basic feature that is nice to have — one user was actually surprised to hear this feature didn’t already exist. Another user explicitly expressed
desire for automatic saving, and undo/redo is a stepping stone for that feature. In itself, undo would be useful for deletions, because sometimes students accidentally delete the wrong item and would need to recreate it.

4.1.3.6 Diff Game Versions

Description
Currently, a user can view all of the different saved versions of his game using the Restore button on the game’s game page. This feature shows the changes made between those saves.

Example
- Sally can see that someone has added a Witch between the most recent version of the game and the previous version.

Feedback
Users responded fairly positively, expressing that it would be useful for helping facilitators or adult designers restoring a previous version of the game or, if two people edited a single game at the same time and overwrote each other’s work, determining whose changes would be easier to recreate. However, one user stressed that kids would probably find this too complicated to use it, or it could initiate a restoring “edit war” in which students continuously try to restore the game to their own work. None of the users ended up ranking this feature very high (or forgot to rank it entirely), suggesting that they did not actually think this feature would be too useful.

4.1.3.7 Comments upon Saving

Description
This allows users to optionally add a few notes when saving the game file

Example
- Sally writes “added a Black Cat” upon saving the game. This will show up under the Restore feature on the game page.
Feedback

Users liked that this would help children learn and practice the important skill of documentation, but felt that it would be difficult and time-consuming to enforce. They were also worried that it would be hard to display the comments in a clean manner. However, it could be helpful for people working on the same game. This idea also sparked one user to suggest allowing comments in the scripts. However, as with the previous feature, users ended up ranking this very low.

4.2 Finalized Suite of Collaboration Tools

Based on the results of the interviews, five of my suggested features were very well-received, and I implemented all but one of them as part of my thesis work. I also worked on a few features that were not part of the original seven I presented to the users. These were designed to solve problems that came up during the interviews. I received positive feedback on the remaining two original suggested features as well, however, both were ranked comparatively low. I did not implement these, although due to the positive response from users, I believe they would be useful additions to the Editor in the future.

4.2.1 Features to Facilitate Version Control

The following features are designed to help users create, view, and restore versions of a game. They are described in detail in Chapter 5.

Section 1: Improved Save Experience

After learning that children — and even adult designers — have difficulty remembering to save the game frequently, I made several improvements to the designer’s experience with saving a game.
Section 2: Improved Conflict Handling

The Editor checks for updates not only every five minutes but also when the designer saves the game. Upon seeing the conflict message described in Chapter 3 Section 4, the designer is given a fourth choice: Save As. With this, she can save the game with her changes as a new game and without overwriting Designer 1’s work. There are now two games, and both designers’ changes are saved.

Section 3: Undo/Redo

Designers can undo and redo all types of changes, from adding a new agent, to relocating a region, to deleting a script, to unchecking a checkbox. In itself, the undo feature is helpful for changes such as accidentally deleting something or dragging to resize a region. Once the Editor offers autosave, being able to undo and redo changes is important in ensuring that the changes saved are actually desired.

Section 4: View Changes

Designers can view all of the edits they made since they last opened the game. Although originally meant to be shown upon receiving a merge conflict message, a list of changes is a useful feature to have at all times. Designers can view all their changes since opening the game instead of since the last save, as originally described to users. This would also help designers decide how much they want to undo or redo to return the game to the desired state.

Section 5: Revision History

While the previous Restore feature allowed users to view and restore a game’s previously saved versions, this feature was difficult to find and inefficient to use. I added more visible methods to visit a game’s revision history and improved the interface for viewing and restoring versions.
4.2.2 Features to Facilitate Sharing

The following features are designed to help users reuse existing games or parts of a game without having to recreate everything from the beginning. They are described in detail in Chapter 6.

Section 1: Remix Games

Designers can create a copy of an existing game and edit or experiment with this new copy. While this feature previously existed, it was difficult to use and explain, and one user explicitly asked for a simpler remixing interface.

Section 2: Look Inside Games

This feature allows a designer to view a game without creating a copy of the game. While not a feature that came up in the user interviews, it helps designers by allowing them to copy from a game without having to remix it. It also provides an elegant one-step method for viewing a game’s source code.

Section 3: Copy/Paste

Designers are able to copy a region, agent, role, beacon, or script from one game and paste it into any TaleBlazer game of their own. This allows designers to choose parts of a game to reuse instead of remixing an entire game. Designers can also piece together sections of different games. Used with Save As, this provides an efficient way of combining changes.
Chapter 5

Tools for Version Control

Successful large projects require a straightforward and efficient method of managing changes to its documents, or version control. This becomes even more necessary when multiple people are adding, modifying, and deleting parts of the project at the same time. This chapter describes a suite of version control tools for the Editor.

5.1 Saving Experience

Proper version control is dependent on designers frequently saving their work. By keeping the last-saved version of the game up-to-date, designers can prevent losing a large amount of work if they accidentally leave the Editor or run into conflicts. This also allows them to experiment with the game and easily discard those changes without deleting earlier work that they would have wanted to keep.

Interview subjects mentioned that children often forget to save their work. For a designer working on a game individually, this tendency is annoying because it could result in having to recreate his work. For a designer who is collaborating with others, this could be disastrous if they are editing the same game at the same time. The conflict message mentioned in Chapter 3 Section 4.1 is triggered only after one designer saves: the longer the designers delay saving, the more changes are made before they are notified of the conflict, and the more work is lost as a result.

Because saving the game is such an important part of version control and effective
collaboration, I began my thesis work with improving the interaction that takes place between the designer and the Editor upon saving the game.

### 5.1.1 Modifying the Save Button

The Editor displays a visual reminder to save the game by adding a dotted red border around the Save button when the designer edits the game (see Figure 5-1).

![Save Game](image1)

![Save Game](image2)

Figure 5-1: The change in the Save button upon editing the game.

In the previous Editor, this did not happen reliably. For example, the Save button remained unchanged when the designer made edits in the Script Editor. That is, if the designer created a new script or updated an existing script, the Save button would not be modified.

The Editor now reliably modifies the Save button whenever the designer makes a change in the game. In addition, the Editor now keeps track of a "dirty" variable that is set whenever the designer changes the game, and this variable is cleared when the game is saved. The state of the Save button depends on the state of this variable, as does an exit game alert described in the next section.

### 5.1.2 Exit Game Alert

The Editor reminds designers to save their game via a pop-up alert, shown in Figure 5-2, when they navigate away from the game, i.e. going back in the browser, closing the tab, or closing the entire browser. However, the behavior of this prompt was — to a typical TaleBlazer designer — rather erratic.

Sometimes the alert would appear when the designer didn’t make any changes that needed to be saved. One common example is a designer opening the game and exiting it immediately, upon which the alert would appear.

50
Figure 5-2: Designers receive this alert when they are about to exit an unsaved game.

Other times, the alert would not show up when expected. One designer, knowing that I was working on improving the save UI, explained how she had been working on a game and accidentally closed it when she was cleaning up the tabs in her browser. She ended up losing a lot of work, and mentioned, “it was weird, doesn’t the game usually display a pop-up when I exit?”

Because the alert is meant to prevent designers from accidentally losing their work, it should appear when, and only when, there is a possibility of this happening. If the alert shows too unnecessarily, as in the first case, designers could become annoyed and immediately click “Leave this Page” out of habit. Worse, they could choose to prevent this page from creating additional dialogs and never see the alert again, even when necessary. If the alert does not show up when the designer expects it, as in the second case, it no longer acts as a reliable safety net.

My solution to this problem was simple but quite elegant: when the designer exits the game, the Editor alerts the user if and only if it is “dirty”. With this design, the alert reliably and unintrusively shows up only when it is desired and expected. This also ensures that the pop-up is consistent with the state of the Save button as well.

5.1.3 Keyboard Shortcut

Finally, many text editors allow users to save their work using the keyboard shortcut “ctrl-s” (“cmd-s” on Mac OS). On a normal browser web page with default settings, such as the previous TaleBlazer Editor, this shortcut produces a pop-up that allows the user to download the webpage’s html and image files. This latter option is almost
never what TaleBlazer designers wants when they press “ctrl-s” in the Editor. Now, this command saves the game instead; designers can save from their keyboard in addition to clicking the physical Save button.

Using the keyboard shortcut will behave exactly like pressing the Save button unless the conflict dialog is open. In this case, the keyboard shortcut does nothing and the designer needs to physically choose the option to save the game and overwrite the other changes. This prevents a designer from accidentally choosing to save and overwrite from pressing “ctrl-s” out of habit.

5.2 Conflict Handling

When two designers work on the same game at the same time, they will run into conflicts. As mentioned in Chapter 3 Section 4.1, the previous Editor had poor conflict management: it would always end up overwriting someone’s work, and there was up to a five-minute time period in which it could do so silently. This section describes the steps taken to fix these issues.

5.2.1 Conflict Dialog on Save

As mentioned previously, the TaleBlazer Editor checked for a more up-to-date version of the game every five minutes. If two designers were working on the same game at the same time, the previous TaleBlazer Editor would display a conflict message within five minutes of Designer 1 saving. This resulted in a five-minute window in which Designer 2 could save her changes and silently overwrite Designer 1’s work without alerting anyone. To prevent this, the Editor now checks for a more up-to-date version when Designer 2 clicks save (in addition to the five-minute time) and displays the conflict message if one is found. This way, Designer 2 is alerted of the change and is presented the choice of discarding her own changes in addition to being allowed to overwrite Designer 1’s changes.
5.2.2 Save As

In a situation of conflict, it is unsatisfactory to simply alert Designer 2 and require her to choose between two conflicting versions of the game, because one of the designers will be losing work no matter the choice. The desire to keep both designers’ version of the game prompted the Save As feature. Upon receiving the alert, Designer 2 is given the additional option to save the game as a new game. Upon choosing this feature, Designer 2 is redirected to a new game, which has only her changes and none of Designer 1’s changes. Designer 1’s game (the original game) will also be unaffected. This idea is loosely based off of the branch feature in Git, as described in Chapter 3 Section 3.1.

![TaleBlazer](image)

Figure 5-3: The very first version of the conflict dialog which provided buttons for each choice. It did not explicitly offer the choice to do nothing.

While relatively simple to implement in the back end, Save As was tricky to present to the designer in the front end. The conflict dialog went through three major design stages and several smaller iterations in between. The wording for the dialog needed to be clear enough to convey what each of the options entailed and concise enough for the designer to read.

As mentioned in Chapter 3 Section 1.2, the Jupyter Notebook conflict message presented its user with buttons along with the corresponding options, requiring less work on the user’s part. Upon adding the Save As feature, I also included buttons to perform each option, as seen in Figure 5-3.
Jupyter explicitly presented the ability to "cancel", or simply exit the dialog without choosing any of the choices. In a later iteration of the Editor's conflict dialog, "do nothing" was made an explicit option as well. To aid the designer in choosing an option, the buttons were also color-coded based on severity of consequence. This was based off of Jupyter’s design as well.

With Save As added as a fourth option, color-coding helps distinguish the choices from each other, as seen in Figure 5-4. The dialog presents the choices in order of decreasing desirability, and the colors reflect this order as well. In most cases, the Save As feature is generally the least harmful choice because neither Designer 1 nor Designer 2 loses his/her changes. Should Designer 2 choose to overwrite changes, it would be better to lose her own changes than to overwrite someone else’s work. Finally, Designer 2 can choose to do nothing and continue working, which would increase the severity of the conflicting versions, and only result in an even larger problem to be solved later.

![Figure 5-4: The dialog was updated to include “Do Nothing” as an option with its respective button. It also uses colors to distinguish the choices and give the user a sense of which is the safest option.](image)

As described in Chapter 3 Section 1, conflicts are caused by one of several reasons. One of the biggest challenges was to write a message that was specific enough to explain the problem to the designer while staying general enough to be able to
encompass all of the conditions. For example, it may make sense to Designer 2 in our example to read that “Another user has made changes to the game and saved before you”, which is similar to the message presented by Confluence (Chapter 3 Section 1.3). However, in another case, a single designer could have opened an unshared game and edited it in two places. He would be alarmed that “another user” was able to access his game even though that “other user” is actually himself. The terms “your changes” and “save my copy” would also be confusing because both sets of conflicting changes are made by the same designer. The final conflict dialog, shown in Figure 5-5, distinguishes the two sets of changes by defining “changes here” to be the ones made by the designer faced with the dialog (Designer 2), and “changes elsewhere” to be the saved changes made by either another designer or the same designer in another location (Designer 1).

![Conflict Dialog](image)

Figure 5-5: The finalized conflict dialog. This version has the most understandable wording and is the most visually appealing.

While Save As is generally the safest option because it does not overwrite any work, it is not the best option in every case. Save As will always result in two separate games. If two designers are collaborating on a game, delay merging their changes, and continue working on the separate games, it will only become more difficult to combine their changes down the line. Save As is more useful in the short term, allowing both designers to save their work without anyone losing their work, but conflicts should be mitigated as soon as possible.
5.3 Undo/Redo

Undo is a feature that is offered by most products, allowing users to delete the last change they made and revert the project to a previous state. This command is often paired with redo, which reapplies the last change that was undone. When interviewing TaleBlazer users, they were surprised to hear that the TaleBlazer Editor did not already offer this feature and felt that the ability to undo and redo a change is “nice to have”. While it is generally difficult to drastically affect a game with a single edit, children have accidentally deleted an entity or a script and regretted the choice. Scratch (mentioned in Chapter 3 Section 2.1) offers an “undelete” option to undo deletions specifically. With the previous Editor, the designer would need to recreate an accidentally deleted item.

While in itself undo/redo does not affect version control very much, it is a necessary stepping stone for an autosave feature. With autosave, the Editor will save automatically at constant intervals, regardless of whether the designer requests to save or not. Thus, it could save some changes that the designer does not want to keep. For example, a designer could open a game, experiment with a new feature, and be unsatisfied with the result. Without autosave, the designer can close the game without saving to abandon the experimental changes. However, with autosave, these changes are automatically saved and the designer would need to go through the game’s revision history to recover the last desirable draft. Implementing undo/redo before autosave is important because it allows the designer to easily undo all of the unwanted changes and return the game to the desired state.

5.3.1 Functionality Design Decisions

While the general idea of undo is fairly straightforward, it is not immediately intuitive to the user what will happen upon undoing. The “Safety” lecture in MIT’s User Interface Design and Implementation course (6.813/6.831) provides a number of design decisions to consider when implementing undo [11]. This section discusses the design decisions made when implementing undo for the TaleBlazer Editor.
5.3.1.1 Defining “Undoable” Changes

As mentioned earlier, a single edit in the TaleBlazer Editor will not usually dramatically affect the game; in fact, most edits — such as checking a checkbox or renaming an agent — are easily undoable without an undo feature. There are a few exceptions: for example, it is much more difficult and time-consuming to reverse accidentally deleting an agent or dragging to resize a region.

However, undo/redo in the TaleBlazer Editor is not meant to be a main feature by itself, but rather a stepping stone for the autosave feature. With this in mind, the designer should be able to revert to a previous version of the game in case the Editor saves undesirable changes. Because of this, nearly every change made by the designer is “undoable”, including toggling checkboxes. While designers looking to uncheck a single checkbox will likely not use the undo button, they will find the undo option helpful if the checkbox is part of a large set of changes to revert.

5.3.1.2 Defining the Action Stream

In some products, the change to be undone depends on the user’s focus at the time. A product’s action stream can be:

1. **global** — in which the product keeps a single list of the user’s changes. Undo will always revert the last change made in the product

2. **restricted** — in which different lists of the user’s changes are kept for different sections of the product. Undo will revert the last change made in that section.

The TaleBlazer Editor keeps a global action stream: undo is designed to revert the last change made, no matter where the designer is focused (e.g. a designer in the Player tab can undo a change in the Agents tab).

Unfortunately, html inputs keep track of their own action stream. Designers can revert the last change in an input by pressing the keyboard command “ctrl-z”. However, the Editor’s undo/redo stream is accessed through respective buttons (user interface design decisions are discussed in the next section). This could result in
different behavior depending on how the designer decides to undo. For example, if the designer performs the following actions:

1. Rename an agent from “Agent1” to “Professor”

2. Create a new agent

3. Focus on the first agent’s name input box

4. Undo

If in step 4, the designer presses “ctrl-z”, this would change the first agent’s name from “Professor” back to “Agent1”. However, if the designer clicks the Undo button, this would remove the newly created agent.

The Editor does not currently offer undo and redo through keyboard shortcuts, so these two methods are separated and consistent: pressing the Undo button will undo the last global change, while pressing “ctrl-z” will undo the last change made within the textbox. However, if the Editor does allow keyboard shortcuts for undo and redo in the future, as discussed in Chapter 7 Section 1, it is important to consider what would happen if the designer presses “ctrl-z” in step 4.

5.3.1.3 Undo Model

The two main types of undo models are linear and nonlinear. With a linear undo model, users can undo only the most recently-completed edit, and must do so before they are able to undo earlier edits. A nonlinear undo model allows users to delete an arbitrary edit from the undo list [12].

The latter is difficult to implement and complicated to reason about. In addition, it is unlikely that such a feature would appeal to a child (or even adult) designer, especially in the context of undo/redo for autosave. Hence, the Editor follows the linear undo model, although a nonlinear model could be useful in the future, as discussed in Chapter 7 Section 1.
5.3.1.4 Redo Model

The simplest redo model is flip-undo, which reverts a single level of undo. With this model, redo is implemented by undoing the last change that was undone; that is, performing undo twice leaves the game unchanged. With this model, there is a single Undo button that toggles the game between two states and the user is restricted to undoing a single change before getting caught in a loop [13]. Most modern products have separate buttons that allow several layers of undo and redo. This allows the users more freedom because they can revert and redo several changes instead of just the last one. Hence, this is the model used by the Editor.

Many products that allow multi-level undo/redo clear the redo stack when the user makes a new edit, rendering any previously redoable edits unrecoverable. Less commonly, some products, such as the Vim text editor, make use of a branching model that allows the user to redo the original change if they accidentally make a new change upon undoing [14]. However, the branching model is complicated to implement and difficult to use. As a result, the Editor clears the list of redoable edits once the designer makes a new edit.

5.3.1.5 Limits

The Editor starts tracking changes once a game is opened. Designers often open a game and work on it for several hours at a time or leave the game open in the same tab for several days. Without an imposed limit, this undo list would grow to extreme lengths, consuming a lot of memory. It is also unlikely that a designer will want to repeatedly click undo to return the game to a specific state. Thus, it makes sense to limit the amount of edits that can be undone. Adobe Photoshop’s undo limit is 20 by default, though users can modify the settings to allow up to 1000 changes [15]. Many of the 2007 Microsoft Office products — including Word, PowerPoint, and Excel — allow up to 100 [16].

The Editor limits the undo history to a fairly arbitrary size of 50. When I explained the situation to a few of the users (in an interview described in Appendix C) and
asked them what size they expected, their answers ranged from 3 to 20, which is much smaller than the actual limit. It is also unlikely that a user will want to undo more than 50 actions. This number can be easily changed in the backend if designers find that this does not provide enough freedom or keeps track of too much data and slows down the Editor.

5.3.2 Implementation

Every change made by the designer is tracked with relevant data to recreate the state of the Editor before the change was made. When tracking a change, the Editor takes note of the entity that was changed (if any), classifies the type of the change, records necessary data to revert the change (e.g. if an agent was renamed, it needs both the old name and the new name), and takes note of metadata needed to recreate other parts of the environment described in the next section.

The Editor’s backend follows an event-driven model: changes to the Editor dispatch events, and parts of the Editor listen for specific events to update themselves accordingly. For example, if a designer renames an agent from “Agent1” to “Professor”, any scripts that reference this agent by name need to update to reflect this.

Initially, it may seem that the undo list should be updated when an event is fired. However, the act of undoing a change also fires events, and this act of undoing is added back into the undo list. With the example above, a designer clicking undo would change the agent’s name back to “Agent1” and clicking undo again would change the name to “Professor”. This produces the flip-undo model mentioned in the previous section, which is not the desired behavior.

Instead, each change made by the user is tracked at one of four points:

1. **Immediately before the change is dispatched**

   **Example:** Deleting an agent

   **Reason:** To undo this change, the Editor needs to be able to recreate the agent. It must keep track of data belonging to the agent such as its name, script, and actions. The agent’s data is deleted along with the agent, so this data must be
preserved before the deletion happens.

2. **Immediately after the change is dispatched**
   
   **Example:** Adding a new agent
   
   **Reason:** To undo this change, the newly added agent is deleted. The Editor deletes a specific agent based on its id, which is not assigned until after the agent is created. Thus, the required data does not exist until after the agent has been added to the Editor.

3. **Both immediately before and immediately after the change is dispatched**
   
   **Example:** Renaming an agent
   
   **Reason:** The Editor reverts this change by setting the agent’s name back to its original value. Thus, it needs to keep track of the agent’s previous name before it was renamed and its new name after it is renamed.

4. **After several related changes**
   
   **Example:** Moving a script
   
   **Reason:** As mentioned previously, the Editor relies on the shared ScriptBlocks library to render scripts. When the designer moves a block, the ScriptBlocks library treats this as three different events: 1) deleting the block from the original position, 2) moving the block, and 3) adding a block in the new position. The Editor combines these three changes into a single change to track so that undo returns the block to the previous position instead of merely removing the block from the new position.

5.3.3 **User Experience**

The Editor offers multi-level undo and redo through separate buttons. This section discusses the design choices made in shaping the designer’s undo and redo experience.
5.3.3.1 Information Scent

As evident from the design decisions that needed to be made, designers might not always be able to predict what will result from undo. Hence, it is important to convey as much information as possible to designers to keep them oriented in the complex Editor interface.

A simple and effective solution is adding hover text to the buttons that describe what change is being undone, as shown in Figure 5-6. The View Changes feature described in the next section (Chapter 5 Section 4) also provides a more complete undo and redo history.

Figure 5-6: A designer hovering over the Undo or Redo buttons will see hover text detailing the change that will be undone or redone.

5.3.3.2 Metadata to Recreate the Environment

Undoing a change is designed to revert the game to the previous state. To aid the designer, the editing environment is reverted to the previous state as much as possible as well. This section describes metadata that is tracked along with the designer's changes in order to recreate the Editor's environment when the edit was made.

Focused Tab and Focused Entity

As mentioned in the Tour of the Editor in Chapter 2 Section 2.2, the Editor is open to one of its six tabs at any point in time. In the Map, Agents, Player, and Beacons tabs, the designer is focused on a region, agent, role, or beacon respectively. The rest of the Editor displays information based on the focused entity.

When tracking a change, the Editor takes note of the tab and, if applicable, the focused entity. Upon undoing or redoing the change, the Editor switches to the same
tab and focused entity as when the change was made.

For example, the designer could rename agent from “Agent1” to “Professor”, which involves being focused on that agent (see Figure 5-7). If he switches to the Map tab then click undo, he will be taken back to the Agent tab, with Agent1 focused.

Open Dialogs
Some parts of the Editor are edited using pop-up dialogs. These include:

1. An agent’s bump settings — These determine under what conditions a player encounters an agent. These settings are set in a pop-up dialog in the agent’s dashboard, or in a general settings pop-up for all beacon agents.

2. The game’s description and game image — These are edited through a normally hidden pop-up form in the navigation bar, as shown in Figure 5-8.

Undoing or redoing a change made in a pop-up dialog will automatically reopen the dialog.

Save Button
As mentioned in Chapter 5 Section 1.2, the Save button receives a red border when the game is modified, and this border disappears once the game is saved. The Editor tracks whether or not it is “dirty” before a change is made in order to properly reflect whether undo (or redo) brings the Editor to a saved state. For example, a designer is working on a game and saves; at this point, the Save button does not have the red border. The designer then creates a new region, dirtying the Editor and modifying
Pick the game name, image, and description to identify this game when browsing or searching on the TaleBlazer website.

This is a game to help students learn about the knapsack problem commonly taught in algorithms classes. Students must maximize the value of items they put in their backpack while keeping the total weight of... To allow other players to play this game, give them this game code: galupvb

Figure 5-8: Designers use this popup dialog to update the game’s description and image.

the Save button. Upon undoing this change, the Editor is back to a saved state and the red border is removed from the Save button.

Figure 5-9: The Save button receives a red border upon changes to the Editor. An unmodified Save button indicates that the game in the current state has been saved.

5.3.3.3 Navigation Bar

As mentioned in the Tour of the Editor (Chapter 2 Section 2.2), a navigation bar resides at the top of the Editor on every tab. Because undo/redo functionality is across all tabs in the Editor, their corresponding buttons need to be visible from each tab and are located in the navigation bar next to the Save button. The navigation bar was further updated to include a View Changes button (see Chapter 5 section 4).
5.4 View Changes

Prior to starting my thesis, I proposed several possible additions to the TaleBlazer Editor, one of which was “display changes”. This was designed to help designers make a choice when they receive the conflict dialog by allowing them to view what changes they had made to the game since they opened it. While originally this feature was meant to be available only upon receiving the conflict dialog, it is a useful tool to have at all times. The ability to view all changes made is also useful for undoing and redoing multiple changes.

5.4.1 Changes List

Both Adobe Photoshop and Paint.NET are image editing softwares that offer remarkably similar History panels that display the changes performed by the user. As the user edits the image, his actions are added to the list with the most recent one highlighted in blue. Upon undoing, the highlight moves to the last action performed on the current state of the image, and all actions after the highlighted one — the next actions to redo — are faded out and italicized. A screenshot of the History panels offered by the two image editing softwares is shown in Figure 5-10. Both images are from the Adobe Photoshop [17] and Paint.NET [18] sites, respectively.

![History panels](image1.png)

Figure 5-10: The History panels found in (left) Adobe Photoshop and (right) Paint.NET

Because space is valuable in the Editor, the designer’s list of changes is hidden.
by default. A View Changes button was added to the navigation bar, and clicking it reveals the list, as shown in Figure 5-11. The list’s design was greatly influenced by the look of the History panels found in Photoshop and Paint.NET: the most recent change is highlighted in blue and the next changes to redo are italicized and highlighted in a darker color. The list displays up to six changes at once (fewer for edits with multi-line descriptions), and once the list is longer, it becomes scrollable. Once the designer is finished viewing the list, he can hide it by clicking the same button, which is now labeled “Hide Changes”.

![Figure 5-11: Designers can click the View Changes button to see the list of edits that they have made since opening the game, and next changes to undo and redo](image)

Adobe Photoshop defaults to a linear undo model but allows users to request a nonlinear model. Depending on the model, clicking on an item in the History panel will either:

- (linear model) Undo or redo all changes between the highlighted change and the selected change, or
- (nonlinear model) Undo or redo the selected change only.

Paint.NET, on the other hand, does not do anything when users select an action. As mentioned in the discussion of Undo/Redo (Chapter 5 Section 3), the Editor follows a linear undo model; thus clicking a change in the Editor’s list should either undo/redo all changes up to the selected item or do nothing.

Initially, I felt that Photoshop’s method of allowing a designer to undo or redo several edits at once is more efficient, however, Lisa was concerned that designers...
might accidentally click on an item and panic upon finding all of their changes disappeared. I asked three different users what they would expect to happen and interestingly, they all expected the change to set the point for the next undo. That is, they expected to be able to undo deletion of the Laptop agent in Figure 5-11 by clicking on the “Delete agent Laptop” item and clicking Undo. They also expected this to undo the deletion only (i.e. it would not rename the region, etc.)

To prevent accidental surprises, clicking a change currently does nothing. Users felt that this was a safe option and were okay with having to click Undo multiple times in order to delete a change further up the list. However, they also felt that displaying the changes in a list suggests that nonlinear undo is available, and believed that it would be a useful option in the future (see Chapter 7 Section 1).

5.4.1.1 Navigation Bar

View Changes, like Undo and Redo, is accessed through the navigation bar.

![Navigation Bar](image)

Figure 5-12: The navigation bar of the previous Editor. It is located at the top of the page and is present in all tabs.

At the left of the navigation bar are two buttons that navigate the designer to other areas of the TaleBlazer site (My Games and New Game). At the right is a textbox to edit the game’s name and the button to display the game’s image and description popup. In the center are five buttons to help the designer build the game:

1. **Emulate** — This runs the game within a browser-based emulator.

2. **Tutorials** — This opens up a side panel that provides documentation, tutorials, and a glossary.

3. **Summary** — Designers can download an html file listing the agents, roles, and other important information about the game.
4. **Error Check** — This alerts the designer of possible errors, e.g. a region that is never visited in the game.

5. **Save Game** — The Save button stores and updates the TaleBlazer database with the contents of the game.

With the addition of the Undo, Redo, and View Changes buttons, the navigation bar becomes slightly overwhelming. To be less intimidating, the navigation bar’s buttons are divided into three main categories:

1. **Navigation** — As before, the two buttons that navigate to different parts of the TaleBlazer site (My Games and New Game) are located on the left.

2. **Checking** — At the center of the navigation bar is a group of four buttons to help the designer check if the game is designed properly. The tutorial button is placed on the left because designers new to TaleBlazer are most likely to use it. The remaining buttons are Summary, Error Check, and Emulate, in that order, which are more likely to be used once a game is complete.

3. **Editing** — Located to the right are four buttons that the designer will use the most often while editing the game: View Changes, Undo, Redo, and Save. These buttons are located within a box that has a lighter background.

A screenshot of the new navigation bar is shown in Figure 5-13.

Figure 5-13: (top) The navigation bar for the previous Editor (bottom) the new navigation bar with View Changes, Undo, and Redo buttons.
The navigation bar is located at the top of the Editor, so previously when the designer scrolled down the page, it disappeared from view. On most computers, an agent’s bump settings, actions, and traits require scrolling to be accessed; when the designer is editing these sections, he would not be able to view the changes list, undo, redo, or save the game without having to scroll back to the top of the page.

To solve this problem, the box containing the four editing tools is “sticky”, meaning its relative position in the page does not change when the designer scrolls. Designers can also click and drag this box to any position within the page, as seen in Figure 5-14. Dragging the box back to the navigation bar will “dock” the tools back into the bar. Figure 5-13 shows the box in docked position.

Figure 5-14: (top) The editing tools are “sticky” and remain at the top of the page when the designer scrolls down (bottom) The toolbox can be dragged to any desired location below the navigation bar.
5.5 Revision History

When speaking with TaleBlazer users about possible features to introduce to the Editor, I asked them for features that they felt would be helpful. One facilitator of Global Kids suggested an “undo button to revert the game to its last working version”. He explained that sometimes a child designer would have an idea to add to an existing game. While initially things seem to be going well, this idea eventually breaks the game. Because the designer had been saving his progress, he can’t just exit the game to discard his changes. The facilitator wanted a way to easily remove all changes related to this new idea. Unbeknownst to him, the previous TaleBlazer Editor already offered this feature. I built upon this existing feature to make it more accessible to designers and more efficient to use.

5.5.1 Previous Restore Feature

The previous TaleBlazer Editor offered a Restore feature, which allowed the designer to view and restore any saved version of the game. This feature was accessed through the game’s game page, which is where the designer can edit the game’s settings (see Figure 5-15). A game’s game page is accessible only to the owner of the game.

A designer can navigate to the game page through one of two ways:

1. By visiting “My Games”, which displays all of the games that belong to the designer, and clicking on a specific game’s Game Page button, or
2. Visiting taleblazer.org/game/gamePage/[gameId]

The second method makes use of a game’s game id, which is one of two unique identifiers for the game. It is used mostly within the TaleBlazer server to retrieve information about the game. The other unique identifier is the game code, which is a short string of letters that TaleBlazer designers give to players in order for them to play their game. This code is entered into the TaleBlazer smartphone app to download the game. Designers are much more familiar with the game code than the game id.
Figure 5-15: The game page for a game with game id “4593” and game code “galupvb”.

Clicking on the Restore button within the game page displayed a dialog with a list of the game’s saved versions. As seen in Figure 5-16, each item in the list shows the time it was saved (in UST), the game’s name at the time, and actions that can be taken. The actions are empty until the designer hovers over a version, at which point it displays two options:

1. **Restore** — This overwrites the game with the corresponding version and closes the dialog, leaving the designer on the game page.

2. **Editor** — This refreshes the page to show the Editor open to the corresponding version of the game. The designer can edit and make changes in the Editor, and upon saving, this edited version will become the most up-to-date version of the game.
5.5.2 Updated Restore Feature

To improve the usability of the Restore feature, I added new methods to access the game’s revision history and updated the user interface for viewing and restoring game versions.

5.5.2.1 Accessing Revision History

The biggest problem with the previous Restore feature was that it was difficult to find. The designer does not frequently visit the game page when editing a game; on the other hand, he spends the majority of his time working on a game in the Editor.

As mentioned in previous sections, the top of the Editor contains a navigation bar that is present across all tabs. This navigation bar contains a button to opens a popup that allows the designer to edit the game’s image and description as well. This popup also reveals the game’s game code which the designer must know to play and share his game. This dialog now includes a link to view the game’s revision history below the game code, as shown in Figure 5-17.

This link only appears for a game that has history and belongs to the designer. That is, the link is hidden if:

- The game is a new game and has not yet been saved, or
- The designer is viewing another designer’s game. The Look Inside feature that makes this possible is described in Chapter 6 Section 2.
Figure 5-17: (left) Designers can click on the right arrow next to the game name to reveal the popup to change game settings. (right) The right arrow is replaced with an edit icon, and the settings include a link to the game’s revision history.

Like the game page, access to a game’s revision history page is restricted to the game’s owner; another designer visiting this page will receive an error.

The designer can also directly view the revision history for a specific game by visiting taleblazer.org/restore/[gameCode]. A game’s restore url is determined by its game code instead of the game id, as the game page is. The decision to use the game code to identify games in user-facing features was made when implementing Remix, which is described in the next chapter (Chapter 6 Section 1). A more thorough discussion of the usability and privacy advantages of using the game code versus game id can be found in that section.

In addition, the Restore button on the game page now redirects to the new revision history page. The designer now has three different ways to find the Restore feature.
5.5.2.2 Displaying Revision History

The previous restore feature was not only difficult to find but also frustrating to use. The list displayed only a UTC timestamp and the game's name. To view the version itself, the designer needed to open it in the Editor. If the version was not the one the designer was looking for, he would need to return to the game page and try again. Once he found the desired version, there was no way to restore the version from the Editor, and he would once again need to revisit the game page in order to restore the version.

One product with a simple and efficient feature for restoring previous versions is Google Docs, which is described in Chapter 3 Section 4.3. The new Restore feature, shown in Figure 5-18, is modeled after Google Docs’ Revision History feature.

Figure 5-18: A game’s past versions is now viewed and restored on a separate revision history page instead of a pop-up dialog on the game page.

A list of the game’s versions is displayed on the left side of the Revision History page, identified by the timestamp – which is converted to the user’s local time zone
– and game name at the point at which it was saved. Clicking on a version reveals a link to restore that version.

Selecting a version also displays a read-only view of the Editor on the right, with the game’s description above. In this Editor, all inputs (e.g. textboxes and dropdowns) and buttons (except the buttons to open dialogs e.g. bump settings) are disabled. The toolbar in the script editor is removed, and the designer can not move or edit blocks, nor can he access the ScriptBlocks right-click menu to duplicate, cut, copy, or paste. Because the designer can not edit anything, the navigation bar, script drawer, and tutorials button have also been removed to save space and present a clean interface.

Because the Restore page attempts to display a vast amount of information in a limited space, the list of versions, game description, and read-only Editor are sized to fill the browser window as large as possible and all dynamically resize with the browser window. Each of the three sections is scrollable if its information does not fit within its assigned size.

With this new interface, the designer can easily view several different versions of the game and restore a specific version without leaving the page.
Chapter 6

Tools for Sharing

In order to effectively collaborate with one another, designers should be able to efficiently share their work. With the previous Editor, two designers working on separate computers could only combine their changes by recreating their work in a single game. This was the case for the children in the Global Kids program. Alternatively, a designer may want to reuse a portion of one game inside a new game. He, too, would need to recreate the relevant parts. Recreating work that already exists is frustrating, inefficient, and prone to error. This chapter describes tools in the Editor to make reusing games and parts of games simpler and more efficient.

6.1 Remix

As mentioned earlier, TaleBlazer game designers can share games by making them public, or remixable. Another designer can make a copy of a remixable game and add her own edits. The other designer saves the game to her own account, and the game's original designer does not see these edits. During an interview with one of the users, I learned that the previous process of remixing was not user-friendly.
6.1.1 Previous Process

As mentioned previously when discussing Revision History (Chapter 5 Section 5), a game has two unique identifiers:

1. **game id** — This is used mostly by the server to retrieve a game’s data and source code. Designers are not explicitly given the game id, but it can be found by opening the game in the Editor, which will redirect to a url of the form taleblazer.org/create/[gameId].

2. **game code** — The game code is the key to downloading the game in the TaleBlazer app, so designers are very familiar with it.

The previous remix process required a designer to remix a game through its game id. Designers would visit a url of the form taleblazer.org/game/gamePage/[gameId] to find a Remix button, as seen in Figure 6-1. This method was unintuitive, complicated, and error-prone.

Figure 6-1: The previous method of remixing a game. Users visit a url based on the game’s unique game id, which in this case is 4593.
6.1.2 Updated Process

The TaleBlazer server now offers a remix landing page. Designers can access this page under the games tab in the TaleBlazer homepage next to the options to view their games or create a new game. They can also directly go to the memorable URL taleblazer.org/remix. Both methods are simple, intuitive, and less prone to mistakes than using the previous URL.

![Figure 6-2: The new remix landing page can be accessed through two easy methods: (left) via a memorable url, or (right) through the games tab.](image)

From here, a designer can specify the games she wishes to remix by entering the game code. This brings up information about the game, such as its description and its owner, as seen in Figure 6-3. If this is the game that the designer was looking for, she can click the button to remix the game. Otherwise, the designer can look for another game by simply entering a different game code.

With this interface, it is possible to enter an invalid game code. This could happen if the game code does not belong to an existing game or the game’s owner disabled remixing. In these cases, the landing page displays an error message with relevant feedback, as shown in Figure 6-4.

This feature was pushed to production toward the beginning of my thesis work. Figures 6-3 and 6-4 are screenshots from the production Taleblazer. The games tab and remix page were later redesigned to allow for the Look Inside feature, described in the next section (Chapter 6 Section 2).
6.1.3 Advantages of the New Interface

The new Remix process has advantages over the previous one because it is simple, makes it easier to share games, improves the TaleBlazer Server’s internal consistency, and addresses a privacy concern.

6.1.3.1 Simplicity

The previous URL to remix a game was long; as a result, facilitators sometimes made errors when writing it out on the board, and children sometimes typed it incorrectly into the browser. To address this concern, designers now remix a game by first visiting a general landing page and then specifying the game they would like to remix. By breaking the process into simpler and more intuitive steps, designers are less likely to make mistakes at any one step.
In addition, TaleBlazer end users will now associate a game with a single unique identifier — the game code. Instead of having to recall both the game id to remix and the game code to share or download a game, designers only need to remember the game code, allowing them to do both.

6.1.3.2 Collaboration

Previously, a designer would not be able to remix a game unless she knew the game’s id, which can only be obtained by communicating with the game’s original designer. Now that games are remixed through their game code, players have the ability to search for any game that they were able to download with a game code. If a player particularly enjoyed playing a TaleBlazer game, she can remix that game using the same code she used to download it. This way, she can explore the source code and add her own experiments, making for a great learning opportunity. Game designers who do not want players to be able to see a game’s source have the option to disable remixing for the game.

6.1.3.3 Consistency

Offering the remix feature through a remix landing page also improves the TaleBlazer server’s internal consistency for the game page. As mentioned in the Revision History section (Chapter 5 Section 5), the game page is where a designer can edit settings for a game that he owns, such as restoring a version or making the game remixable. The gamepage url is the same as the url to remix the game; as a result, a designer visiting the old remix url for a game of his own will see the page shown in Figure 6-5. This page has a Copy button, which is entirely equivalent to remixing the game to both the end user and the server. A designer using the game page to administer his own game may want to make a copy of the game, however, a designer accessing this page through the remix url may be confused because there is no Remix button.

With the new Remix landing page, designers will see no difference between remixing their own games and remixing someone else’s game other than the fact that they own the game, as seen in Figure 6-6. Now, a designer would visit the game page url
only to edit settings for a game, so he would not be confused to see the Copy button instead of a Remix button.

Figure 6-5: A designer visiting the previous remix link will see the game’s game page, which has no Remix button.
6.1.4 Privacy

As mentioned previously, the game id is used internally by the TaleBlazer server to retrieve a game's source code. When a designer edits a game, he is taken to a url of the form taleblazer.org/create/[gameId]. Previously, any other designer could view the game's source code by visiting this url as well.

New TaleBlazer games are assigned game ids in sequentially increasing integers. This means that visiting the url for a random number between 1 and the maximum game id will likely reveal the source code for a TaleBlazer game. Someone looking for a specific game could run a brute force search by visiting the url for each number.

This is potentially concerning because TaleBlazer games are by nature based on location, and children will often want to create games located around their neighborhood. A stranger might visit the source code of a game and access sensitive information about the creator of the game. The solution to this problem is two-fold.

With the new Remix page, the end user never needs to know a game's game id because she is remixing through the publicly distributed game code instead. In addition, when a designer remixes a game, the server creates a new game id when copying the game, and the game id of the original game is never shown in the process. If a game is not remixable and does not belong to the designer, the page returns an error; therefore access to a game's source code through remixing is restricted to only remixable games and games that belong to the designer.

In addition, access to the source code of a game that does not allow remixing is now restricted to only the game's owner. With this in place, someone visiting every possible game url would be able to see only remixable games and his own games.

With the new remix page and privacy settings in place, designers who do not want others to access their game's source code can simply disable remixing for the game.
6.1.5 User Feedback

I received feedback from two of the three users I interviewed in Chapter 4, and their responses were positive. They felt that it was easier to not only remix a game but to explain the remixing process as well. They also liked using the game code to remix, because TaleBlazer users are already familiar with game codes in order to download and share games for mobile downloads.

6.2 Look Inside

Scratch is another blocks-based programming language, mentioned in Chapter 3 Section 2.1, and it offers a See Inside feature. TaleBlazer now provides this feature as well. This allows designers to view a game’s source without remixing it first; that is, the designer can view the game without adding it to her list of personal games, and the server will not need to create and store a copy of a game that will be unused. This feature allows designers to view games without cluttering their games list and the TaleBlazer database. In addition, copying an entity, described in the next section (Chapter 6 Section 3), requires a designer to be able to open a game, and this feature allows him to do so without needing to remix the entire game.

The Look Inside option can be found inside the Remix page. To search for a game, designers follow nearly the same steps as remixing:

1. Visit the remix landing page.

2. Enter the game code.

3. Press the Look Inside button.

There is also a “Look Inside” option in the games tab (as seen in Figure 6-2), which redirects the designer to the same remix landing page. Since this landing page now serves a dual purpose, it is re-titled “Find A Game” and has a new interface, as seen in Figure 6-8. In addition, designers can directly look inside a game by navigating to taleblazer.org/look/[gameCode] in the browser.
Find A Game

Enter the game code of the game you would like to remix or look inside:

galupvb

Find Game

Knapsack

This is a game to help students learn about the knapsack problem commonly taught in algorithms classes. Students must maximize the value of items they put in their backpack while keeping the total weight of all their items under a certain threshold.

Owner: lindaw16
Location: 3 Ames St, Cambridge, MA 02142, USA

Figure 6-8: The updated remix landing page, retitled “Find A Game”, allows designers to either remix or look inside a game.

The purpose of the Look Inside feature is to allow designers to view games that they do not own; it does not make sense for designers to use this feature for a game of their own when they already have view and edit access. For this reason, designers searching for their own game in the remix landing page will see an Edit Game button instead, as shown in Figure 6-9. This opens the game in the Editor.

Figure 6-9: A designer searching for his own game will see an Edit button instead of Look Inside.

A designer looking inside someone else’s game can still make changes to that game inside the Editor. In this case, the Save button is replaced with a Remix button, as
seen in Figure 6-10. Pressing this button will create a new game with the designer’s changes and leave the original copy of the game unmodified.

![Image of Remix Game button](image)

Figure 6-10: A designer looking inside a game that he does not own will see a Remix button instead of Save.

A designer viewing or making changes inside an outdated game will not see the conflict message when the game is updated. For example, if the following steps take place:

1. Designer 1 is working on a game.
2. Designer 2 looks inside Designer 1’s game and makes changes as well.
3. Designer 1 saves.

Designer 2 does not get the conflict message at this point. If Designer 2 presses the Remix Game button, the Editor will create a new copy of the game with only her changes and none of the changes made by Designer 1 after the version opened by Designer 2.

### 6.3 Copy/Paste

While it is useful to be able to copy and build upon an entire TaleBlazer Game as remix allows, designers may desire to reuse only certain parts of a game. In addition, remix does not provide a way to piece together parts from several games into a single game.

Copying and pasting is offered in almost any popular text-editing application — users have a body of text in one area that they would like repeated or relocated to another area; instead of rewriting each word, users can select the text, make a copy,
and paste it in the desired location. This allows users to reuse a small section of the original text and combine sections of texts from several sources.

Hence, copy/paste is the final feature in the suite of tools included in my thesis. A designer can copy one of the four main entities — regions, agents, roles, or beacons — and paste it in any TaleBlazer game. In addition, the designer can copy and paste a section of a script as well.

Since the TaleBlazer Editor is not text-based, it is not immediately intuitive what parts can be copy/pasted or what exactly happens upon pasting. There were several interesting cases to consider, each of which had multiple possible solutions. Even after the functionality was decided, there was the question of how to present the copy and paste features. This is the first feature offered in the TaleBlazer Editor that allows designers to combine parts of different games. While copy/paste is simple in concept, it becomes complicated when it comes down to the details.

This section assumes a basic familiarity of the TaleBlazer Editor. Chapter 2 Section 2.1 gives a detailed tour of the Editor that provides necessary background information. This section begins by discussing the backend design decisions made in implementing copy/paste, then goes over some details that were important to ensure proper implementation. Finally, it describes the process of designing the interface for copy and paste.

6.3.1 Functionality Design Decisions

When making design decisions, I first used my own judgment as a novice TaleBlazer user to imagine the expected response for a designer who has about as much experience with the Editor as I do. I received feedback for these decisions from Lisa and Judy, who are experienced TaleBlazer users, and for some of the cases even we were unable to fully agree upon an intuitive behavior for pasting an agent. Lisa and I discussed these difficult cases with two of the TaleBlazer users I interviewed as part of my research before starting my thesis work. Each interview lasted one hour over a remote video conferencing software with screen-sharing capabilities, and a summary of the interview protocol can be found in Appendix B.
6.3.1.1 Depth of Copy

One of the first design decisions that needed to be made was what copying an entity entails; for example, an agent has several layers of information, including but not limited to:

- name, description, and image
- location, icon, and bump settings
- actions and traits
- scripts

It is not immediately clear how much of this information should be copied along with the agent. Lisa, Judy, and I agreed that a pasted agent should stay as true to the copied agent as possible, but we wanted to hear what our end users thought as well.

I asked the users to imagine that they were in the situation similar to one that Global Kids is familiar with: the children have been working together on a single game. They have separated into three groups to create a beginning, middle, and end for the game, and each group is working inside a separate TaleBlazer game. How would the facilitator use copy and paste to combine these three games into one single game?

One of the users mentioned copying agents and their scripts separately, but when I prompted him further and asked if he did not want the scripts to be copied along with the agent, he exclaimed, “if the scripts were included, it would be even better!” Both users agreed with our initial thoughts that everything in the original agent should be recreated in the pasted agent.

Disagreement arose when it came to the subject of whether a copied region would include all of the agents in the original game that are located within it. One user worked extensively with indoor regions and beacons, and felt that it would make sense for a region to come with all of the agents (and beacons as well). The other user, a facilitator at the Missouri Botanical Garden, often uses custom images (e.g. a map of the Botanical Garden) on top of the satellite image. One of her biggest challenges
is positioning the custom image so that it lines up with the embedded Google Maps. She felt that when she copy and pastes a region, she would want just the map of the region without the associated agents.

In the previous TaleBlazer Editor, designers have the ability to duplicate a region, which does not include the original region’s agents. I decided that it would make the most sense to the end user if copying a region, agent, player, or script and pasting within the same game has exactly the same functionality as duplicating it. Hence, copying a region does not include the agents.

6.3.1.2 References to Non-Existing Information

While pasting an entity within the same game is guaranteed to produce the same result as duplicating the entity, pasting into a different game produces some interesting cases. One common case is: a designer copies something that references information found in the first game, and this information is not found in the second game. This section will discuss the decisions made for non-existent values in script dropdowns, assets, agent locations, and bump settings for beacon agents.

Populating Dropdowns in Scripts

A script may contain game-specific references, and can be copied in one of two ways:

1. A portion of a parent entity — For example, the designer copies the agent that the script belongs to. As discussed in the previous section, this includes all of the agent’s information, including the script.

2. Individually — As mentioned previously in Chapter 3 Section 1.3, ScriptBlocks offers its own delete, cut, copy, and paste through a right-click menu.

The script can then be pasted either within the same game or into a different game.

Figure 6-11 is an example of a script made of four blocks:

- The first block is a control block and determines when the script is executed.
When player bumps Professor

Figure 6-11: A script belonging to an agent named “Professor”.

In this case, it is when the player encounters the agent that this script belongs to, named “Professor”.

- The second block references the action “Say Hello” that belongs to the Professor.
- The third block references a Laptop agent that is also found in the same game.
- The final block switches the game to the Map tab which displays the region in which the player is located. This is found in all games, so it can be populated directly.

Depending on how the script was copied and where it is pasted, the middle two blocks are not always guaranteed to be populated with their original values.

If the script was copied with the Professor agent, it will be pasted as part of the agent as well. Since copying an agent copies all of its data, the copied Professor agent is guaranteed to have the Say Hello action. However, if the script was copied individually, it could be pasted into an agent, or other entity that has scripts, that does not have the Say Hello action. In this case, the Say Hello action referenced in the dropdown is replaced with a “???” label, as seen in Figure 6-12.

If the script is pasted within the original game, the Laptop agent will almost always exist. An edge case would be the designer deleting the Laptop agent before pasting, described later in this section. However, the script could be pasted into another game which does not have the Laptop agent. In this case, the Laptop agent referenced in the dropdown is replaced with a “???” label, as seen in Figure 6-12.

Replacing references to non-existing entities with the “???” label is consistent with the Editor’s behavior if a designer deletes the entity. For example, if the designer
deletes the Say Hello action or the Laptop agent, the script will update to replace the labels with "???".

Figure 6-12: The script’s dropdown menus will be populated differently depending on how it was copied and where it is pasted.

Pasting an entire entity and pasting an individual script is done through different interfaces, which is described in the next section. This means that it is easy to tell whether the script was copied as part of an entity or alone. However, it is harder to determine whether the destination game is the same as the original.

As mentioned in previous sections, TaleBlazer games have a unique game id. Initially, it seems that comparing the game ids would be a reliable method of whether or not the script was pasted within the same game. However, there is an edge case: games are not given game ids until they are saved, and all new games have the same invalid id of -1. Hence, this method would falsely deduce that two different newly created and unsaved games are the same.

In addition, pasting within the same game still does not guarantee the existence of all selected entities. For example, a designer could copy the script, delete the Laptop agent, and then paste the script in the same game. The script should no longer be prepopulated with Laptop because it no longer exists.

Instead of trying to determine whether or not the script is pasted into the original
game, the Editor goes through each selected value in the script’s dropdown and checks if it exists in the game in which the script is pasted. Each option in a script’s dropdown has a label and an id, and the script keeps track of the selected id. The label is usually the name of the item (e.g. “Laptop”), and the id is the item’s unique id in the game.

When recreating the script upon paste, the Editor reviews the options for the dropdown and checks whether the originally selected option exists by searching for an option with the same label and id. If none is found, the Editor assumes that the originally selected option does not exist in the new game and replaces the selected option with a “???” label.

This method takes care of the two corner cases described above, but introduces a new corner case of its own: the script could be pasted into a different game that happens to have an option with an identical label and the same id, in which case the Editor will erroneously prepopulate the dropdown with this option. For example, if the script in the example were pasted into a game that happens to have a Laptop agent with the same id as the Laptop in the original game, the Editor will assume these two Laptops to be the same, even though they could be completely different in every other way (e.g. description, actions, etc.). However, this case is very rare, as it is unlikely for two entity in different games to share the same name and the same id.

Finally, there is an edge case that is not covered by either method: even if the designer copies and pastes the Laptop agent into a new game before copying and pasting the script, the new Laptop will likely have a different id than the original. The Editor will not recognize that these two agents are equivalent and fail to prepopulate the dropdown.

While the current method of populating the dropdowns for a pasted script is not perfect, it is reliable in the vast majority of cases. An alternative method of populating dropdowns would be to require a different method of determining equality between options. This would entail keeping track of additional data for the selected option in the original game, and comparing this data instead of ids. However, this complicates the implementation (other than name and id, there is no field that is common to all
agents, actions, etc. in the Editor), and would increase the comparison time. This is undesirable, especially when the current implementation performs well for the vast majority of cases.

Assets
TaleBlazer game designers can upload images to a game’s file for regions (custom maps), agents, and roles. Within a game, the Editor keeps track of a ref count for each image, which tracks the number of entities that use the image. An accurate ref count is vital to having images work properly — i.e. if the image of a copied agent already exists within a game, its ref count should increase by one upon pasting it; otherwise, it should be added to the game with a ref count of one. Some series of actions made for very baffling results when I originally failed to properly keep track of the ref count:

Case 1:
Steps to reproduce

- Copy the only agent in the game that has an apple image.

- Delete this agent and save the game.

- Paste the agent in the same game and save the game. This agent has the apple image.

- Refresh the page.

Result
The pasted agent no longer has the apple image.

Case 2:
Steps to reproduce

- Copy the only agent in the game that has an apple image.

- Paste the agent in the same game. Both agents now have the apple image.
- Delete the pasted agent and save the game.

- Refresh the page.

**Result**
The original agent no longer has the apple image.

Saving the game removes all images with a ref count of 0 from the game's files. Because the ref count was originally not properly maintained upon pasting the agent, deleting either the original or the pasted agent in the game decreased the count to 0, and the image was deleted from the game files upon saving the game. Therefore, when the game tried to search its files for the image upon being reopened, the image could not be found.

**Location of an Agent**
Every agent in the game must have a valid location. This location could be in a specific area of a region, or the agent could be specified as a Clue Code agent. For the latter option, the agent is bumped if the player enters a valid Clue Code when playing the game. This feature was added to encourage students to look up and be aware of their surroundings instead of staring at their phone screens during the entire game.

For agents that are located in a region, a problem arises when they are pasted into a different game. A region is defined by latitude and longitude boundaries that are accurate to several significant digits. It is not intuitive how many significant digits should match in order for two regions to be considered the same. Hence, to the Editor, two regions that are from different games are different by default. As a result, agents that are pasted into a different game will always have an invalid location.

Like script dropdowns, the Editor checks whether an agent’s location is valid by searching the game for a region with the same name and id as the original. With this method, agents that are pasted within the same game will have the same location if the region still exists, but if the designer had deleted the original region before
pasting, the pasted agent will have an invalid location as well.

I discussed the possibility of warning the user of invalid locations with Lisa, Judy, and my users. We felt that while a pop-up warning would be useful for the user the first few times, it would become annoying after popping up every time, especially for children. A less intrusive response that is still intuitive is to place agents with invalid locations in the upper left corner of a user-defined Start Region, which is the default location for new agents.

**Beacon Agent**

One way for a player to encounter an agent indoors is to walk within range of a Bluetooth beacon. These agents are known as beacon agents, and must be associated with a beacon configured via the beacon tab. When an agent is pasted into another game, its associated beacon needs to be in the game as well.

Unlike other TaleBlazer entities such as regions, beacons can easily be checked for equality. Bluetooth beacons are uniquely identified by their UUID, major id, and minor id. If two Bluetooth beacons have the same three ids, then they are equal.

Upon pasting a beacon agent, the Editor checks if its associated beacon already exists within the game. If it doesn’t, it displays the prompt showing in Figure 6-13 asking if the user would like to add the beacon. If so, the Editor will add the beacon to the game and set the agent to be bumped via the newly added beacon.

![Warning](image)

**Warning**

Professor(copy) is bumped via a beacon that is not yet in this game. Would you like to import the beacon to this game?

Yes No

Figure 6-13: The designer is asked whether he would like to import the beacon along with the agent he is pasting.
6.3.1.3 Common Traits

An agent's trait can either be specific to the individual agent or common to all of the agents in the game. One interesting case appears when an agent with a common trait is pasted into a new game.

Originally I felt that it would be strange to add the common trait to all of the other agents in the new game — as a user, I would not expect pasting a new agent to affect the existing agents. Because of this, I concluded that all traits in a pasted agent should belong to that agent only. However, upon speaking with Lisa, she mentioned that common traits are most often used for keeping score in roles, and it would not make sense to have a common trait be specific to a single role.

The users I spoke with also had strong differing opinions. Without hesitation, the Global Kids facilitator responded that he would want the common trait to be added to all of the agents in the new game, whereas the Missouri Botanical Garden facilitator felt that she would feel “weirded out” if the trait appeared in other agents as well.

In the end, we decided that for the large majority of cases, the common trait should stay common. That is, upon pasting the agent with the common trait into the new game, the Editor will add this trait to all other agents as well. If the trait already exists (e.g. the designer has already pasted this agent into the game), it will not be added a second time.

For the case where the designer does not want the common trait, he can simply delete it in the Editor.

6.3.1.4 Multiple Entities vs. Single Entity Pasting

The copy/paste functionality exists for regions, agents, roles, and beacons. One design decision discussed was whether the designer should be limited to copying only one type of entity at a time. I originally felt that a designer should be able open a game and copy a region, an agent, a role, and a beacon. The designer could then open a second game and paste a region in the Map tab, switch to the Agent tab and paste
an agent, and so on. This way, the designer would only need to go back and forth between the two games once instead of four times. However, Lisa and Judy felt that it made more sense to be able to copy and paste only one entity at a time.

I presented my users with the following situation: *Suppose you have two games, and you are copying some agents and roles from one game to the other. After copying an agent, you get distracted momentarily. When you return to TaleBlazer, you’ve forgotten exactly where you were. You click the Paste button in the Player tab. What would you expect to happen?*

Both users expected an error message of sorts and neither believed that their last copied role would appear. To follow the users’ expectations, the Editor allows only one entity to be copied at a time. If a designer copies an agent and clicks the Paste button in the Player tab, the Editor displays an error message, as seen in Figure 6-14.

![TaleBlazer](image)

You copied a(n) agent but you can only paste roles here!

Figure 6-14: The designer receives a useful alert upon pasting an entity in the wrong tab.

### 6.3.2 Implementation

As mentioned in Chapter 3 Section 1.2, TaleBlazer uses local storage to transfer data between games. When an entity is copied, the Editor stores the following information:

- **Type** — This notes whether the entity is a region, agent, role, or beacon. This is used to prevent the designer from pasting an entity in the wrong tab.

- **Basic Information** — This includes the entity’s name and other fields depending on the type of entity. For example, an agent’s information includes its location and a beacon’s information includes its UUID.
- **Asset** — This is information regarding the entity’s image. The basic information tracks only the id of the asset, so this is necessary when pasting the entity in a game that does not already have the asset.

- **Scripts** — These are the scripts associated with a copied agent or role.

- **Actions, Traits** — These are the actions and traits of a copied agent or role.

- **Beacon** — This is a beacon agent’s associated beacon.

As mentioned in the previous section, the Editor checks the game to see if the entity’s asset (if it has one) already exists in the game. It does so by searching through the game’s assets for one that has the same url and size. If it finds one, it assumes that the game already contains the entity’s asset. The Editor checks the game for existing common traits (if the agent has any) in a similar manner: two traits are considered equal if they have the same name, type (whether it belongs to an agent or role), and value.

One important note is that the actions and traits need to be created before the scripts in order for the dropdowns to populate correctly. As mentioned before, the Editor keeps track of only the id of a dropdown’s selected option; however, when actions and traits are recreated upon pasting, they receive new ids that are very unlikely to match their original ids. Dropdowns that reference the agent’s own actions and traits are updated with their new ids.

In addition, actions can be associated with a script. For example, the Professor’s action “Say Hello” can trigger execution of certain blocks, as seen in Figure 6-15. Since scripts are not recreated until after actions and traits, the Editor needs to go through each script action and connect it to the appropriate script after the scripts are added.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Content</th>
<th>Visible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Say Hello</td>
<td>script</td>
<td>Greeting</td>
<td>yes</td>
</tr>
</tbody>
</table>

![Greeting](say"Hello!")

Figure 6-15: The “Say Hello” action is connected to a script named “Greeting”.
6.3.3 User Interface Design Decisions

As mentioned previously, the TaleBlazer script editor is built on the ScriptBlocks library which offers duplicate, cut, copy, and paste through a right-click menu. I was able to update the existing paste function, which was previously limited to be within the same game, to allow pasting across games.

![Right-click menu for ScriptBlocks](image)

**Figure 6-16:** ScriptBlocks offers a right-click menu to copy and paste blocks.

However, copy and paste are completely new features for the rest of the Editor, and creating a visible, clean, and intuitive user interface for these features proved to be a complicated task.

Originally, I believed that a right-click menu like the script editor would be simple and consistent. However, Lisa and Judy were concerned because right-click is not visible to the user, and TaleBlazer designers would be unlikely to find this new feature.

This situation came up again when discussing copy/paste functionality when Judy suggested that I allow the designer to paste an agent directly onto a region. Unbeknownst to me, right-clicking on a region brings up a context menu allowing designers to create and relocate agents, as seen in 6-17.

To see whether the users knew about this feature, I asked them to describe how they would add an agent to a specific place in the map. Neither of them described using the menu because they did not know of its existence either. Both users and I are very experienced with the Editor, but we did not know about an incredibly useful feature because there was nothing indicating that regions had a right-click menu.

In order for copy/paste to be useful to the designer, a good user interface design is just as important as its functionality. While the right-click menu is fairly prevalent for copy and pasting (e.g. ScriptBlocks and many text editors), copy and paste need to be as visible as possible in the Editor.
Figure 6-17: Right-clicking on a region displays a context menu that allows the designer to create a new agent, paste an agent, or move an agent to the clicked location. The paste button is disabled if the designer did not copy an agent.

There were a few other concerns to address in designing the interface, and I designed three different UI mockups that addressed these concerns between them. I showed these mockups to users of varying experience with TaleBlazer and other technologies. Based on their feedback, I combined different parts of the mockups and finalized the copy/paste interface.

6.3.3.1 Concerns

As mentioned in Chapter 2 Section 2.3, the Map, Agents, Player, and Beacons tab contain a ribbon displaying a list of all of the game’s entities of the type.

Each entity in the ribbon appears on its own tile. This tile shows the entity’s image and name, and allows the designer to delete or duplicate the entity, as seen in Figure 6-18. Clicking on an entity’s tile populates everything below the ribbon with the entity’s information, such as its name, actions, and scripts. Designers must select an entity to edit its settings but can delete or duplicate an entity without selecting it by directly clicking the buttons on the entity’s tile.
Figure 6-18: The previous interface for the agents ribbon, with six agent tiles. The Professor is currently selected but the designer can directly delete or duplicate any other agent.

Because the tile is very small, the buttons are not labeled until the designer hovers over them. While the delete icon is fairly intuitive, a designer new to the Editor may not recognize the plus icon to mean duplicate without the hover text.

At first glance, a copy button for an agent seems to fit nicely right below the duplicate button. However, it is not obvious where the corresponding paste button would go. While copying is specific to a single entity, pasting is specific to the entire tab. This also brings up three main concerns:

1) **Redundant Duplicate Button**
   As mentioned in the previous section, duplicating an entity is the same as copying the entity and pasting it in the same game. One question that came up was whether having copy/paste obsoletes the duplicate feature. Duplicate requires half the steps but having both clutters the interface, creating a trade-off between efficiency and cleanliness of design.

2) **Obscure Icons**
   The agent tile is not large enough for button labels, so the icons need to be intuitive enough for designers to recognize their functionality at a glance. However, as seen in Figure 6-19, interfaces do not agree upon universal, distinct, or intuitive copy, paste, and duplicate icons.

   - **Duplicate** — While TaleBlazer uses a plus icon to signify “duplicate”, several other interfaces use either two sheets of paper with an arrow from one to another
or just two sheets of paper.

- **Copy** — Some interfaces use two sheets of paper, as described for duplicate. Another common design for copy is an image of a clipboard with a sheet of paper or an arrow to mean "copy to clipboard", however, this design is remarkably similar to some paste icons.

- **Paste** — Many interfaces use an image of glue or a clipboard to signify paste.

![Duplicate, Copy, Paste icons](image)

Figure 6-19: Image search results for duplicate, copy, and paste.

ScriptBlocks offers cut, copy, and paste functionality through buttons in addition to a right-click menu. It uses two sheets of paper to signify copy and a clipboard to indicate paste, as seen in Figure 6-20. These buttons are helpfully labeled.

![Copy, Cut, Paste buttons](image)

Figure 6-20: The ScriptBlocks buttons for Copy, Cut, and Paste

### 3) Unrestricted Pasting

The designer may not realize that after copying an entity, it can be pasted into another game. Previously, only text could be copied in one game and pasted into another, and it is not immediately intuitive that this can be done with entire entities and scripts as well.
6.3.3.2 Existing UI Research

For user interface design ideas, I researched several existing products that allow copy/pasting of non-text items across projects and windows. This section mentions the following:

- **GameBlox** — This is another blocks-based programming platform, introduced in Chapter 3 Section 2.2.

- **Google Drawings** — This product allows users to create diagrams and flowcharts by creating and rearranging icons and shapes.

- **Balsamiq** — This is an application for creating wireframes and mockups.

All of the projects I researched offered at least one of the following four interfaces:

1) **Right-Click Menu**

Like ScriptBlocks, all three products mentioned above offer copy/paste through a right-click menu.

**Pros** — It is a fairly common method for text and draggable items.

**Cons** — It is not visible to the user, and designers who are used to creating TaleBlazer games without copy/paste are not likely to find it.

2) **Keyboard Shortcuts**

The universal shortcut for copy is “ctrl-c”, and “ctrl-v” for paste. On Mac OS, the cmd key is used instead of ctrl.

**Pros** — Like right-click, it is a fairly common method of using the copy/paste feature.

**Cons** — Like right-click, it is hidden from the user. In addition, copying with a keyboard command could behave differently depending on which element of the user interface has the focus (e.g. inside a textbox or on a script).

3) **Menu Bar**

Some applications such as Google Drawings offer copy/paste through a menu bar at
the top of the screen, as shown in Figure 6-21.

**Pros** — For applications with a menu bar, it is common to find copy and paste under the Edit option. It is relatively visible and intuitive.

**Cons** — The TaleBlazer Editor does not already offer a menu bar. Menu bars usually include several options, so it does not make sense to have one with only copy and paste.

![Menu Bar Diagram](image)

Figure 6-21: Google Drawings offers copy and paste through the Edit option in the menu bar.

4) **Toolbar**

Some applications such as Balsamiq have copy and paste buttons in a toolbar, as shown in Figure 6-22. Unfortunately, these icons are hard to understand without hover text. The button with two sheets of paper signifies duplicate, while the two sheets of paper with an arrow is for copy, which is reversed from the images search results in the previous section.
Pros — Of the four methods discussed in this section, the toolbar offers the most visibility.

Cons — It is very difficult to determine what each of the icons represent without hover text, especially for new users.

Figure 6-22: The five left-most buttons of the Balsamiq toolbar are for undo, redo, duplicate, copy, and paste.

Visibility is top priority for the new copy and paste features, and the toolbar is the best way to ensure this. Hence, the TaleBlazer Editor currently offers copy and paste buttons on a toolbar. A right-click menu and keyboard shortcuts would be nice additional ways to offer copy/paste once designers are familiar with it.

6.3.3.3 Mockups and User Testing

I created three separate Balsamiq mockups of the new user interface for the Agent tab. The toolbar on the Map, Player, and Beacon tabs would look nearly equivalent, but would refer to regions, roles, and beacons instead of agents respectively. Across the mockups, I tried different looks and locations for both the existing buttons (New Agent, Delete, Duplicate) and the new buttons (Copy, Paste).

I showed each of these designs to several users of different ages and with a variety of experiences with TaleBlazer. Many users were undergraduate students who were working on StarLogo and GameBlox. While they had little experience with TaleBlazer, they were very familiar with at least one blocks-based programming interface. I also remotely user-tested with two experienced TaleBlazer users over screen-share.

Depending on the user’s knowledge of TaleBlazer, I gave a brief introduction of TaleBlazer’s mission and how a designer would use the Agents tab in the Editor. With each user, I first showed them two copies of one of the three designs and explained that these were two different games, opened in two separate windows. I asked them to think out loud as much as possible while they performed each of the following tasks:
1. Create a new agent (named “Agent2”) in Game 1.

2. Copy the Earth agent in Game 1 and paste it in Game 2.

3. Go back to Game 1 and delete Agent2.

I then asked for their opinions on the look and feel of the interface as well as the ease of performing these tasks. I repeated these steps with the remaining two designs and finally asked for their favorite design and any overall opinions. I rotated through the six possible orders of designs.

The following screenshots of each design are taken during step 2, when the user copies the Earth agent in the first game.

Design 1

Figure 6-23: Design 1 is most similar to the previous Editor, which is above for reference.
This design is the most similar to the previous Editor. The buttons to delete, duplicate, and copy an agent are always present, so users could use those buttons on an agent without selecting it first. The Paste button is below the New Agent button, which was previously located to the right of the Editor above the agent ribbon.

Almost every user tried to select the Earth agent when asked to copy it — even one of the experienced TaleBlazer users — even though this step is unnecessary. Several of the new users did not recognize the copy icon and also asked what the plus icon was for. Users also felt that the buttons appearing on each agent were a bit redundant and cluttered the interface. However, while the new users did not like this interface very much, the Missouri Botanical Garden ranked it the highest because it was the most efficient; it had a duplicate button and did not require an agent to be selected in order to be deleted.

Design 2

![Image of Design 2](image)

Figure 6-24: Design 2 is least similar to the previous Editor.

This design is the most different from the previous Editor. The option to duplicate an agent is removed, and the Delete and Copy buttons are above the ribbon, making up a small toolbar. The New Agent button is unchanged. Most notably, the user needs to select an agent before deleting or copying it.
This interface was voted the best by nearly all of the users I asked because it had a “clean layout”, and, more importantly, was “very intuitive”. Users appreciated that the buttons were labeled. One user mentioned that having the copy and paste buttons above the ribbon better hints at the ability to paste the agent across games, whereas having the copy icon within an agent tile made the copy/paste feature feel more restricted to a single game. Both experienced TaleBlazer users commented on the disappearance of the duplicate icon, but the Global Kids facilitator voted this his favorite despite this absence.

While the new users felt that selecting an agent before deleting it provided a sense of security, the Missouri Botanical Garden facilitator, who has worked with adult game designers and complicated TaleBlazer games, was worried about this requirement. She mentioned that agents with long and complicated scripts have noticeable loading times, and it would be annoying to have to wait for an agent to load before being able to delete it.

Design 3

![Design 3](image)

Figure 6-25: Design 3 is a combination of Design 1 and Design 2.

This final design is similar to Design 1 in that the Copy and Delete buttons are placed on the agent tile, but like Design 2, the Duplicate button is removed and the
agent needs to be selected in order to be copied or deleted. The buttons show up only for the selected agent. The Paste and New Agent buttons are placed next to each other and in the middle of the screen, and the New Agent button is in the same style as the Paste button.

The feedback for this design was very similar to feedback for the corresponding parts of Design 1 and Design 2. Users who saw this interface first also commented that they did not recognize the copy icon and wished it were labeled in the same style as the paste icon. They provided similar nonchalance (or wariness) toward selecting an agent before acting upon it, and the two experienced TaleBlazer users again noticed the lack of the duplicate button. One user commented that the paste button looked strange without a corresponding copy next to it.

6.3.3.4 Resulting Interface

It was interesting to see how users with different experience levels had different expectations and reactions to the interface.

While the new users did not mind — and actually preferred — having to select an agent before deleting or copying it, the experienced users knew that this method would require twice the work and possibly much longer wait time. Lisa brought up a case where an adult programmer had an agent so complicated, the browser would freeze whenever it was selected. In the end, the agent needed to be removed from the game, and this would have been impossible to do if it had to be selected first. This was a case that new users did not think of.

There was also a difference in expectations of technology. Neither of the adult TaleBlazer users I interviewed reacted to the copy icon, and the Global Kids facilitator even remarked that the icons were "fairly universal signs for copy and paste". However, most of the new users did not recognize the icons because they couldn't recall seeing them or using them. When asked to come up with an icon that would mean copy or paste to them, they could not think of anything. Most modern-day products do not require clicking buttons to copy/paste, but rather offer the feature through keyboard shortcuts and right-click menus instead.
Because the copy/paste feature will be new to nearly all TaleBlazer users, it needs to be as learnable as possible. For this reason, the final design, shown in Figure 6-26, looks much like Design 2 because of its large labels for each button. Most users felt that this was the most intuitive and visually appealing of the three.

![Figure 6-26: The finalized agent tab, with large, labeled buttons above the ribbon. Designers can hover over a specific agent (in this case the Apple) to display duplicate, copy, and delete buttons for that agent.](image)

However, this would also require the user to select an agent before copying or deleting it. To make these actions more efficient for the experienced TaleBlazer designer, agent-specific duplicate, copy, and delete buttons appear upon hovering over a tile. These hover buttons do not have permanent labels, but they are in the same order as the labeled buttons above the ribbon and have identical icons. Hovering over the buttons themselves will also display hover text describing what they do. In Figure 6-26, the mouse is hovered over the delete button for the Apple agent while the designer is focused on the Professor agent.

While having the copy and paste feature makes the duplicate feature unnecessary, both experienced users noticed and commented on its disappearance in Designs 2 and 3. Upon doing some research, most interfaces that allow copy and paste (including the aforementioned GameBlox, Google Drawings, and Balsamiq) also have a duplicate option. The previous concern was that having both options would make the interface feel too cluttered, but the final design is space-efficient, allowing room for this button. Having both duplicate and copy/paste also hints that pasting will work across games because duplicate would be completely redundant otherwise.

One user commented that the New Agent button was hard to find because it was located on the right side of the page, but the normal direction of focus in English applications moves from top-down and left-to-right. Thus, the most important and
frequently used buttons should appear in the top left corner. In the new design, the New Agent button is also relocated to the left of the screen, swapped with the detail and overview toggle panel that was previously in its place.

This New Entity button and the Duplicate/Copy/Paste/Delete toolbar look the same in the Map, Agents, Player, and Beacons tabs.
Chapter 7

Future Work

While the Editor now boasts a suite of both new and improved tools designed specifically to help TaleBlazer designers with saving, building, and sharing their games, this suite can still be expanded and perfected. This section suggests some ways to improve the suite of tools I added and also revisits some features that were explored during my interviews with TaleBlazer users prior to the start of my thesis.

7.1 Improving the Existing Suite

While I worked hard to make a user-friendly and production-ready suite of tools, there is always room for improvement. This section suggests some ideas to build upon the existing tools.

Nonlinear Undo

As mentioned in Chapter 5 Section 3.1.3, the Editor currently supports a linear undo model, allowing designers to undo only the most recently completed action at any point in time. However, some applications such as Adobe Photoshop allow nonlinear undo, in which users can undo a specific action. For example, suppose a designer makes the following changes:

1. Delete agent Professor.
2. Create a new agent, Agent1.

3. Rename Agent1 to “Apple”.

4. Add several scripts to Apple.

At this point, the designer suddenly regrets deleting the Professor. However, he has already completed a lot of work on the new Apple agent. With nonlinear undo, he can bring back the Professor without losing his work with the Apple.

Currently, TaleBlazer games can only be edited from a single account, so linear undo does not cause any problems. However, this may become an issue if a future version of the Editor allows multiple users to simultaneously edit a game from different accounts.

The current linear undo model will force the multi-user undo to follow a global model where the Editor reverts the last edit to the game, regardless of who initiated the undo and who made the change. In order to support a local multi-user undo model in which designers can only undo their own actions, the undo/redo model will need to be updated to a nonlinear model.

However, the nonlinear model is complicated by the notion of causality. For example, it is not immediately intuitive what happens if two designers are working on the same game at the same time from different accounts and the following happens:

1. Designer 1 renames Agent1 to “Professor”.

2. Designer 2 deletes Professor.

It is unclear what should happen if Designer 1 clicks undo at this point. Does the Professor agent come back as Agent1? Does nothing happen? What if Designer 2 clicks undo afterward? What if Designer 1 clicks redo after that? It gets very complicated very quickly.

Designing and implementing a nonlinear undo model for the Editor would make for an interesting and rewarding project.
Keyboard Shortcuts

As mentioned in previous sections, keyboard shortcuts are offered by several products. As part of the suite of tools, I implemented “ctrl-s” to save the game. It would be useful to introduce the following shortcuts and commands:

- “ctrl-z” for undo
- “ctrl-shift-z” for redo (some interfaces use “ctrl-y”)
- “ctrl-c” for cut
- “ctrl-v” for paste

The keyboard shortcuts might behave differently depending on where the designer is focused in the editor. As discussed in defining action streams for undo/redo (Chapter 5 Section 3.1.2), html input boxes have their own action streams: that is, when the designer is focused on an html input box, clicking the Editor’s Undo button can lead to a different result than pressing “ctrl-z”. It is important to consider what would happen if the user uses a keyboard shortcut in an area where it is already defined.

Tutorials

The TaleBlazer Editor currently offers tutorials that explain parts of the Editor and teach the designer how to create parts of a game. Designers would benefit from tutorials for some of the new tools that were added and existing tools that were improved in this thesis, such as Save As (Chapter Section 2), Revision History (Chapter 5 Section 5), Remix (Chapter 6 Section 1), and Copy/Paste (Chapter 6 Section 3).

7.2 Expanding the Suite

As part of my research before starting my thesis, I interviewed TaleBlazer users and discussed several features that would improve collaboration between designers. Some features that I did not implement but may be useful are:
Merging Changes

When a game is edited and saved in two places, this feature would automatically merge the changes, asking the designer to manually resolve conflicts if necessary. The users I interviewed felt that this feature would be incredibly useful, but only if it were done well. After implementing the features described in my suite of tools, I felt that having a flawless merge feature would be out of the scope of my thesis work and my time and effort would be better spent on perfecting the features in the suite instead of starting an incomplete merge algorithm. My thesis work involving Saving (Chapter 5 Section 1), Conflict Handling (Chapter 5 Section 2), and View Changes (Chapter 5 Section 4) brings the Editor closer to being capable of a merge feature, and it would be a challenging thesis topic for a future student.

Autosave

Instead of requiring the designer to remember to periodically save their work, the Editor would automatically save the game’s progress. To help designers feel more secure using this feature, my thesis work involved implementing undo/redo as a first step (Chapter 5 Section 3). Frequent saving will bloat the database with several unnecessary drafts of the game and complicate the search for a specific version of the game, so an autopurge algorithm will also be necessary.

Diffing and Comments for Game Versions

A designer can use the new Revision History page (Chapter 5 Section 5) to view several saved versions of a game from a single page. A diffing feature would display the changes made between game versions. A commenting feature would allow a designer to write a short summary of changes when saving and view it from the Revision History page as well. These features were ranked by users to be lower than the ones implemented in my thesis but still received very positive responses.
Commenting on Scripts

After discussing comments on a game version, one user suggested that it would be useful if designers could add comments to their code. Scratch and GameBlox are other blocks-based programming platforms that offer this feature, as seen in Figure 7-1.

![Figure 7-1: Comments for scripts in (top) Scratch and (bottom) GameBlox](image)

Procedures

This user also suggested introducing procedures, a user-defined collection of blocks. These are useful if multiple entities in a game have similar scripts. Instead of repeating these blocks for each entity, designers can call a single procedure block that represents the entire collection. This makes scripts shorter and easier to read. In addition, a designer wishing to make changes need only edit the procedure instead of updating each agent individually. This feature is offered by Gameblox, Scratch, and StarLogo.
Chapter 8

Contributions and Conclusion

8.1 Contributions

The work encompassed by this thesis builds a suite of tools that improve a TaleBlazer designer’s ability to manage his game’s versions, build upon existing games, and ultimately work on a game with other designers. This involved:

- Updating existing features to be more reliable and user-friendly, such as:
  
  - Saving a game
  - Displaying reminders to save an edited game
  - Detecting and handling conflicts in a game
  - Remixing a game
  - Viewing a game’s revision history
  - Restoring a previous version of a game

- Introducing several new features for the Editor designed to help the designer manage a game’s versions, build upon existing games, and collaborate on games with other designers, including:
  
  - Saving a game with merge conflicts as a new game without overwriting any changes
- Undoing and redoing changes made to a game since it was opened
- Viewing a list of changes made to a game since it was opened
- Viewing a game without remixing it
- Copying a script in one game and pasting it in any other game
- Copying an agent, region, role, or beacon in one game and pasting it in any other game

- Designing and implementing intuitive, clean, and efficient interfaces for each of the aforementioned features

8.2 Conclusion

TaleBlazer’s mission is to create immersive learning experiences for children by providing a platform for building and playing educational location-based augmented reality games. These games are often the result of a team effort and involve combining work done by several people. However, the TaleBlazer Editor used for creating these games was previously designed for games created by a single person. This made collaborating on games error-prone, inefficient, difficult, and frustrating for TaleBlazer designers.

The work of this thesis involves building upon existing features and adding new tools to address this problem, focusing on better version control and improved methods of sharing work. These additions and improvements will make it easier and more efficient for designers to work on projects with fellow TaleBlazer game designers.
Appendix A

Preliminary User Interviews

The following is a summarized version of the general protocol for the user interviews conducted as preliminary work for my thesis. It has been condensed to be more concise than the actual interviews, which went into more detail, especially if the user asked clarifying questions. The actual interviews were also slightly modified depending on the user’s prior experiences and his or her responses as the interview unfolded.

Introduction

Hi TaleBlazer User,

I’m Linda, and I will be working with TaleBlazer for my Master’s Thesis until the end of the year. My main goal is to improve sharing and collaboration between multiple TaleBlazer game designers working on the same game in the Editor. Here’s a quick overview of today’s interview. First, I would love to first hear about your experiences with the TaleBlazer Editor as it currently is, specifically when multiple people are collaborating on a single game. I’ll then ask you to suggest some features that you’d like to see, and finally, I’d like to receive feedback on some specific ideas for modifying the Editor to improve collaboration.

Q. (for Global Kids facilitators): You’ve coached children in afterschool programs where an entire classroom of kids collaborate on a single game. Thinking of times
when more than one student was working on a single game file, did you have multiple kids working on the Editor at the same time? Could you explain how that worked?

(for the Missouri Botanical Garden facilitator): You’ve coached both professional game designers as well as children creating TaleBlazer games. How did multiple people coordinate work simultaneously on a single game file in the Editor?

Q. What were some problem they encountered?

(for the Missouri Botanical Garden facilitator): Were the challenges faced by the adults different from the challenges faced by children?

Q. Was there anything that worked particularly well?

Q. Are there any features or ideas that you think would have been helpful to overcome those challenges?

**Feature Feedback**

I would like to introduce some ideas to you, some of which are based off Google Docs (real-time collaborative editing) and, if you’re familiar with it, Git (version control). Most of these ideas are for people editing a game through the same account.

I’d like you to rate these not useful, somewhat useful, extremely useful, and provide any additional feedback. Please also comment on:

- how would you imagine the students using this?

- what problem would this feature solve?

- why?

**Displaying Changes**

Display a user’s edits

- Show what the user has changed since they last saved the game file
• John opens his Halloween TaleBlazer game and adds a Black Cat. Sally opens the same game, adds a Ghost, and saves first before John is able to save. John can see that he added the Black Cat before choosing whether to accept the new version of the game or lose his Black Cat.

Diff game versions

• Currently under “Restore” in a game’s Game Page, users can see all of the different saved versions of their game. This will show the changes made between those saves.

• John can see that he added a Black Cat between the most recent version of the game and the previous version.

Merging Changes

Import Agent (across games)

• A user can copy/paste an agent across game files.

• John decides to create a Thanksgiving game and wants to reuse the Black Cat — he can copy/paste it from the Halloween game without having to recreate it from scratch.

• Sally is creating a Pet Shop game in a different account and also wants to reuse the Black Cat — she can copy/paste it into her game as well.

Merge changes between simultaneous edits (within a game)

• If a game is edited by two users at the same time, currently, the second user to save will need to decide whose changes to overwrite. This will merge the changes, automatically where possible and user-guided where needed.

• With the current Editor, John would be given a message saying that there is a conflict. With this feature, John would see the same message but instead of
having to choose between his changes and Sally’s changes, he can merge their changes. The page will reload, and John will see both the Black Cat and the Witch.

Miscellaneous

Undo/Redo changes

- Allow the user to undo/redo their last action within a session
- If John deletes his Black Cat agent, he can then change his mind and click “undo”, bringing his Black Cat back into the game. He can also “redo” the deletion, removing the Black Cat once more.

Save As

- Save As another version of the game upon save (in case of conflicts)
- John can save his current state of the game as a different version, essentially branching off the main version. There will now be two different versions of the game - one with only the Black Cat, and one with only the Witch.

Comments upon save

- Allow users to optionally add a few notes when saving the game file
- John can write “added a Black Cat” upon saving the game. This will show up under “Restore” in the Game Page.

Conclusion

Q. After discussing these features I suggested, are there any new ideas that you have thought of?
Q. Do you have any additional comments or suggestions? Did something stand out to you (both good and bad)?
Appendix B

Discussion of Remix, Save As, and Copy/Paste

The following is a summarized version of the general protocol for the user interviews conducted to obtain ideas and feedback for the Remix (Chapter 6 Section 1), Save As (Chapter 5 Section 2) and Copy/Paste features (Chapter 6 Section 3). Each question that I asked the user was designed to answer a specific research question, and if relevant, was followed by a few probe questions. The actual interviews slightly differed depending on the user’s experience with TaleBlazer and responses to previous interview questions.

Introduction

Hi TaleBlazer User,

I am currently working on my Master’s thesis, which is focused on improving collaboration between TaleBlazer game designers. We spoke a few months ago to brainstorm some ideas, and I am working on implementing a few of those ideas. Today, I’d like to hear your thoughts on three features: Remix, Save As, and Copy/Paste.
Remix

A new remixing interface is available at “taleblazer.org/remix”.
(If the user needs an example game code: galupvb)

Research Question: Would you use it?
Interview Question: How do you feel about using the game code to remix? Do you think it is useable? Any suggestions or feedback?

Research Question: Does it provide enough, and relevant information?
Interview Question: The page currently shows the game’s owner, description, location. Is there any other data you really want to see (e.g. number of downloads)?

Research Question: Is it intuitive to access this page?
Interview Question: The remix page is also accessible from the TaleBlazer home page. Where would you look for it?

Save As

Research Question: How can we improve this message to make it concise and understandable?
Interview Question: A game is opened and edited in two tabs. The person working in the first tab saves. The second tab will see this message, and will be given four choices.
Interview Question: Do you think this message is clear? Would a child be likely to read it and understand it?
Copy/Paste

I am implementing copy/paste of agents, roles, and regions across games. You can open up two different TaleBlazer games and copy from one game to another.

**Research Question:** How does a user expect to access copy/paste in a TaleBlazer game?

e.g. Should it be necessary to select an agent before copying?

Interview Question: Imagine that you are copy and pasting between two games. Walk me through how you are using it and what you are using it for.

*Probe:* How do you feel about having to select an agent before copying? (currently you can clone an agent without selecting it first)

**Research Question:** What does a user expect and want to happen when they copy/paste? Does copying the region include [a subset of] the agents in that region?

Interview Question: Imagine that you have two games. You add a tutorial region that contains a set of tutorial agents to one of your games. You are happy with this tutorial and you decide to implement the same tutorial in the other game. How would you use copy/paste to do this?

What if in the second game, half of the agents are the same?

What if none of them are the same?

*Probe:* When you copy a region, do the agents come with it?
If yes:
  Do all of the agents that ever appear there come with it?
  Do only the agents that start there come?
If no:
  Explicitly ask if they’d want the agents

**Research Question:** What does a user expect and want to happen when they copy/paste? Does copying an agent include the beacon?

Possible outcomes (that I can think of)
  Warning
  Automatically add beacon
  Provide the option to add a new beacon upon paste

Interview Question: Have you used beacons in games before? (explain if no) You have two games. You add an agent that is triggered by a beacon to one game. You decide to copy this to the second game. Do you expect the beacon to be copied to the second game along with the agent?

*Probe:* When you copy an agent, how detailed does it get?
  Does it include scripts? Checkboxes? Bump settings?

Interview Question: Now suppose you decide to copy a beacon from one game to another. Do you expect the agents triggered by the beacon to be pasted along with the beacon?

**Research Question:** Can you copy multiple different entities at once? i.e. can you copy a region, an agent, a role, and a beacon at the same time?

Possible outcomes
  Error if you copied an agent
  Paste the last map you copied
  Silently paste an agent
  Paste an agent and switch tabs

Interview Question: You have two games. You are copying some agents and roles
from one game to the other. After copying an agent, you get distracted momentarily. When you return to TaleBlazer you've forgotten exactly where you were. You click “Paste” in the Player tab. What do you expect to happen?

*Probe:* Explain your thought process.

**Research Question:** Can we remove the clone button?

Interview Question: Currently there is a “clone” feature for agents, regions, and roles. Do you expect copy/paste within the same game to be different than “clone”?

*Probe:* If no:
  - Would you like to see copy/paste instead of clone?
  - Would you have a preference for one or the other, given both?

If yes:
  - How?

**Research Question:** How can we improve the visibility of right-click menus and features?

Interview Question: You want to add an agent to a specific place in the map. How would you do this?

*Probe:* Did you know you could add agents in the map tab?

If yes:
  - How did you discover this?

If no:
  - How would you make this feature more visible?

**Research Question:** How should we present the copy/paste feature to the user?

(Run testing with mockups over screenshare)
Appendix C

Discussion of Undo/Redo, View Changes, and Revision History

I'm going to ask you a couple quick questions about Restore, Undo/Redo, and View Changes. I will also ask a few user interface questions for a couple other features.

Undo/Redo

The Editor now offers undo and redo made for all changes to a game. This feature will be a stepping stone for implementing autosave, so that a designer can undo any unwanted changes if the Editor automatically saved them. This keeps track of all of your changes, such as deleting an agent, clicking a checkbox, and writing scripts. To prevent this feature from taking up too much memory and making the Editor really slowing, I'm imposing a limit to how many changes it keeps track of.

Q: Off the top of your head, how much should this limit be?

Followup: Adobe Photoshop has a default limit of 20 changes, although the user can change this to keep track of up to 1000. Microsoft Office has a limit of 100. Knowing these numbers, would you change your answer?
**View Changes**

I’m adding the option to allow the designer to view all the changes made since opening the game (limited to the size of undo). This helps the designer remember what they did, and also helps show what undo or redo will do. This is what it currently looks like:

![View Changes interface]

The last change you made is highlighted in blue, and the changes in gray are ones that you made but undid. In this example, the designer made several actions, and he just undid updating the DVD agent’s location and renaming it.

Q: Suppose that you are the designers who made the changes. You suddenly regret deleting the Professor agent. How would you undo deleting it?

*Probe:* (if the user clicks on the action)
What do you expect to happen?
- undo all of the actions
- only bring back the Professor

How would you feel if the opposite happened?
How would you feel if it undoes all of the actions up to the Professor?

Q: One of the changes is “Add action **Action2**”, which is when I created a new action for one of the agents. Do you think it should say what agent it belonged to? (e.g. Add action **Action2** to agent **Lisa**)

Q: How would Rename look?
- Rename action **Action2** of **Lisa** to **Say Hello**
- Rename action **Action2** of agent **Lisa** to **Say Hello**
- Rename **Lisa**’s **Action2** to **Say Hello**
Rename agent Lisa’s action Action2 to Say Hello
Rename action Action2 to Say Hello for agent Lisa

Q: Any feedback or questions about undo/redo or view changes?

Editing Toolbar

As you can see from the View Changes screenshot, the undo, redo, and view changes buttons are next to the save button, and they are all part of the navigation bar. Here is a screenshot of what the navigation bar looks like now.

The four buttons are in gray because the designer will be using them the most when editing a game. This toolbar is "sticky", so when you scroll on the page, the buttons will continue to be in view.

Q: Looking at the screenshot below, do you feel like you can do anything else with them?

Q: Here are two looks for the toolbar. What does each one tell you?
Revision History

TaleBlazer has a “Restore” feature that allows designers to revert a game to a previous version (walk through previous Restore process), and I worked on making this feature easier to find and simpler to use (go through current process).

Q: Currently, this feature is found in the game page, under a button called “Restore”. Other products (like Google Docs) calls it “Revision History”. Another name we came up with is “Earlier Saved Version”. Which one would you think makes the most sense?

Q: I would also like your opinion on what to call each of the individual versions themselves. Some words we came up with were:

  Version
  Saved Version
  Draft
Q: Any comments or questions about the interface as a whole? For example, where would you put the description of the game?

 Probe: Is having the description useful?

Find A Game

I am introducing a new copy and paste feature, and we realized that in order to copy from a game, designers need to be able to open the game in the first place. To prevent them from having to remix every game they want to copy from, I added a Look Inside feature (similar to the one offered by Scratch) to allow users to directly open a game in the Editor. Since this is also done through the game code, we are adding this button to the Remix page. This is the previous interface and updated interface:
Remix A Game

Enter the game code of the game you would like to remix:

galupvb

Find Game

Knapsack

Owner
lindaw16

Description
This is a game to help students learn about the knapsack problem commonly taught in algorithms classes. Students must maximize the value of items they put in their backpack while keeping the total weight of all their items under a certain threshold.

Location
3 Ames St, Cambridge, MA 02142, USA

Remix Game

Find A Game

Enter the game code of the game you would like to remix or look inside:

galupvb

Find Game

Knapsack

Owner
lindaw16

Location
3 Ames St, Cambridge, MA 02142, USA

Remix

Look Inside

The page has been remixed to “Find A Game”
The “Look Inside” button says “Edit Game” if you own the game and takes you to the Editor.

Q: Any thoughts on this updated interface, or the look inside feature?
Bibliography


