CSCI370 Final Report

CSM Gianquitto

Jose Acosta, Brandon Her, Sergio Rodriguez,
Sam Schilling, Steven Yoshihara

Table of Contents

1.0 Introduction

Mapping the Routes of Science is an interactive, web-based game that takes the user on the journey of Ynes Mexia, a famous botanist. The player travels through various regions of South America to deliver an important message to Ynes. However, her exact location is unknown. In order to find her, the user must travel around a topographical map by clicking on various destinations of importance and gather clues that will help him/her find Ynes. Along the way, the user will learn how Mexia navigated through South America as well as what research she conducted while visiting each location.

The goal of this project was to encourage children, specifically children ages 11-14, to take a more active interest in a topics such as science, geography, and history. Our client, Dr. Tina Gianquitto of the Liberal Arts & International Studies department here at the Colorado School of Mines, initially began this project in the spring semester of 2016 with her special topics group. Her class implemented the use of Mapbox (a mapping platform similar to Google Maps) for navigating through South America and had the basic design of one interactive game. The proposal she made for us was to take the work that her students had done and create an educational yet engaging game that tells the story of Mexia's journey.

In order to accomplish this task, we first had to become familiar with the software used by the previous group as well as a number of programming languages that we had little to no experience with. These languages included JavaScript, HTML, and CSS. Once we were able to understand what exactly we were working with, we created the storyline and basic design for the game (mini-games, dialogue, etc.). After we did some research and fleshed out a route from the beginning of the game to the end, we were finally able to begin coding in order to create the game.

2.0 Requirements

2.1 Functional Requirements

- Game should start when user inputs the site passphrase
    - The map should load and start the user at the first point
    - The first clue should appear and be clickable
    - The external information window should be openable by a button

- When the user tries to advance the game, they should be allowed to continue, unless there is an event or mini-game
    - When the user completes the mini-game or event, they should then be allowed to continue

- The clues should appear as the game progresses
    - When moving to a new scene, the clue should be visible as a label on the map
    - When the label is clicked, a popup should show up giving the player a story or choices on what to do
    - On certain parts, the popup should then call the sliding window for a mini-game or additional information

- The game should end when the user reaches the last part of the story
    - When the user reaches the last point and delivers the letter to Ynes, the game should return to the main menu screen

- The external info (sliding window) should operate as such:
    - When not in the middle of a mini-game, the window should show a news feed dialogue (that is also being updated) as well as achievements
    - During a mini-game, the window should display the HTML that corresponds to the specific mini-game at that point
    - When closed during a mini-game, the window should leave the screen without affecting the rest of the game
    - When closed not during a mini-game, the window should not affect the rest of the game

- Mini-games should work as described:
    - For dialogue mini-games:
        - There should be up to three choices that can lead to different dialogue options
        - The buttons will either progress or show a popup telling the player that they have not selected an appropriate option
    - For matching mini-games:
        - When clicked, the picture or term should show that it has been selected by now having a border or changing text color, respectively
        - When matched, the picture and term should show that it has been matched by changing to a grayscale image or changing to gray text, respectively
        - If selected or already matched, the picture or term should not be able to be chosen again
        - A player should not be able to select two items from the same side
        - A continue button should appear after all matches have been made that allows the user to exit the mini-game
    - For click-and-find mini-game:
        - There should be a set of clickable objects within an image
        - When a correct object is selected, the game should notify the player and highlight it
        - When an incorrect object is selected, the game should notify the player with a popup
    - For quiz mini-game:
        - The player should be shown a list of questions with available radio button choices
        - A player should only be able to select one answer from the list of answers
        - The game should respond with some message based on the number of correct answers
    - For boat transport mini-game:
        - The player should be able to select the loads they want to add
        - The player should be warned if the load cannot be added
        - The player can tell the boat to transport, causing the boat to move to the other side
        - The boat should empty and be ready to move afterwards
    - For boat maneuvering mini-game:
        - The player should be able to control the boat
        - The boat should respond when hitting an obstacle

2.2 Non-functional Requirements

- Maintain the interest of the user(s) through interaction with an educational discovery game
- Tell the story of Ynes Mexia's journey accurately
- Have a visually appealing website (design, organization of information, etc.)
- Avoid "damsel in distress" mantra, empowering women
- Should incorporate significant events within the journey and include sufficient details within each section of the game
- Avoid copyright infringements with the content used in the game

3.0 System Architecture

Figure 1 shows the overall system design for our web game. The system itself is not very complex nor does it rely on any major external resources besides Mapbox. The main interaction between the game and player is through the browser. The player will be focused on advancing through the game with interfaces like dialogue choices, clickable buttons, and a map (using Mapbox).

The game itself is located on Luna, a server on the CSM campus. It is accessible on any machine within the school's network via a web browser. On Luna, the game consists of a hierarchy of folders stored under *MappingRoutesOfScience*. The main component is the map and from there it is broken down into mini-games and the main JavaScript that runs the game.

The map is loaded and displayed using a JavaScript file that populates the main HTML file. As the game progresses, another JavaScript file refers to and loads other HTML files that contain the mini-games. A JSON file referenced by the JavaScript contains the story and markers that are displayed on the map.

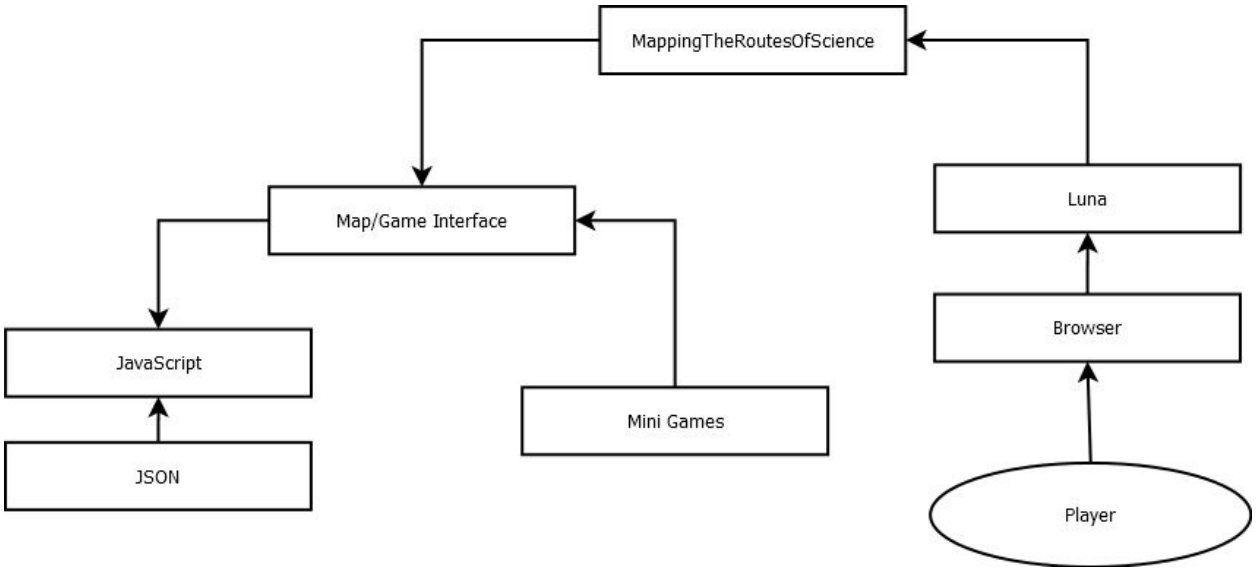

Figure 1: Overall System Architecture

4.0 Technical Design

4.1 Tools

For this project the tools used were:

- HTML
- CSS
- JavaScript - jQuery

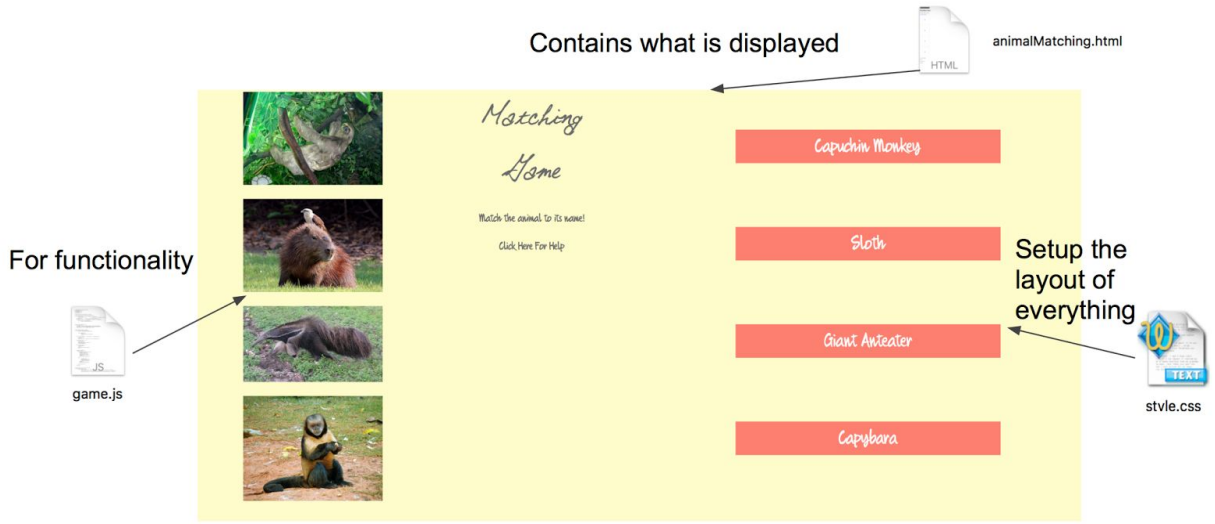

Figure 2: General Idea of tools

4.2 Loading Mini-games

In favor of easy integration of mini-games in the map interface, mini-games were developed independently of the map. Ideally, integrating these mini-games would be a simple call to the mini-game's HTML file, which would refer to its own resources. The first attempt to accomplish this was to edit the map HTML with the javascript `getElementById.InnerHTML` property along with reading in the mini-game's HTML file. This decision was made due to a lack of knowledge regarding the interactions between JavaScript, jQuery, and HTML. Although it did work, there were some graphic and programmatic anomalies, as well as inconveniences when attempting to integrate the mini-games. So, we decided that this method was not ideal for our project. The second attempt utilized the jQuery `load` function. This method correctly loaded all the mini-games except for one (described in the next section). It also allowed the map interface and the mini-games to be independent of each other, aside from references to shared resources due

to our decision to put the mini-games in the slide in panel in the map's HTML (discussed in further detail in section 5.2).

4.3 Matching Mini-game

The idea behind this mini-game was based on the concept of having images on one side with the corresponding name on the other side as seen in **Figure 3**.

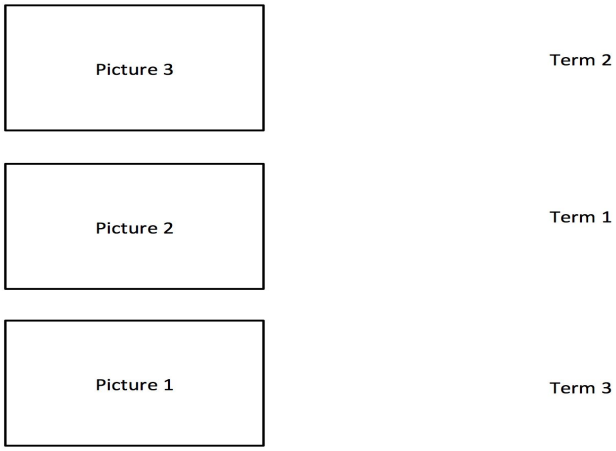

Figure 3: Design of Matching Mini-game

Our first attempt at creating this game was to design a game where the HTML was basically empty. The JavaScript contained code to create individual pair objects, as shown in **Listing 1**, that would be inserted into the HTML as pictures or terms. All that needed to be added was the name of the file and the name of the picture and the JavaScript took care of the rest. This design was successful on its own, but had many integration issues with the main code. Once again, this can be attributed to a lack of understanding the interaction between JavaScript and HTML.

```
/* pair object to store directory to image/term */
var pair = function(id, type, but_id) {
       this.id = id;           // id to check matches
       this.but_id = but_id;          // id for the corresponding html object
    this.type = type;   // if it is a term or definition
    this.state = 2;       // 0 = match, 1 = chosen, 2 = not chosen
    // function to check for matches based on the id -> not button id
       this.is_match = function(pair) {
              return (this.id == pair.id);
       };
};
```

Listing 1: Constructor for a Pair Object

One of the most prominent issues arose from how the HTML was loaded onto the main map. These included problems with the shuffling function and progress saving. The shuffling function would randomize the pictures and terms, but would also duplicate pictures or not allow terms to be matched to their corresponding pictures, thus making the game unplayable. Another issue that came up occurred when a player attempted to exit the game before finishing it. The results would remain programmatically the same, but the game would appear to restart. This by itself would not have been a problem. However, because the main JavaScript needed to be in the main HTML, none of the variables ever got reset. So, when restarting the game, the HTML was reset and the previous selections would not be shown, which made the game difficult to play.

With those complications, the design of the mini-game had to be modified. The HTML is hard coded to be filled with the objects for the pictures and terms. The JavaScript was reduced to a select few functions that are meant to check for matches and update the selected items accordingly. It is more complex now to enter a new object, but it is still possible to create multiple HTML files that use the same JavaScript source. The new design now has the HTML acting as a template that can be duplicated and modified for multiple games, with the JavaScript containing the code needed for basic functionality.

Figure 4 shows some of the results a player can have while playing the game. The cases possible in this mini-game include:

- A match is found
- All matches are found
- Two items on the same side are selected
- An object is already matched or chosen

Figure 4: 1 - Shows an image matched, 2 - Shows an item selected, 3 - Shows a term matched

4.4 Dialogue Engine

In order to support multiple dialogs of arbitrary complexity, we created a simple yet robust engine that supports the creation of new dialogues and modification of existing dialogues. This engine was designed as a sort of pseudo domain-specific language (DSL). In order to create a dialogue interaction, we created a dialogue object using code written in `dialog.js`, and then passed a story to its `start` function. The story is simply a list of objects with information describing each part of the dialogue interaction, including the dialogue text itself as well as some other properties.

Below is an example of the code used to create a simple dialogue interaction.

```
<script src="assets/js/dialog.js"></script>

<script>

var story = [
    { label: "first_question",
        img: 'assets/images/selena.jpg',
        question: "Hi there, my name is Selena. Are you hungry?",
        responses: [
            { m: "Yes I'm famished! What do you have?", next: "label_yes" },
            { m: "No, I just ate.", next: "label_no" },
        ]
    },

    { label: "label_yes",
        p: "You probably shouldn't accept food from strangers..."
    },

    { label: "label_no",
        img: 'assets/images/selena.jpg',
        question: "Okay, I guess I'll eat this all by myself.",
        responses: [
            { m: "Continue", next: "story_end" },
        ]
    },

    { label: "story_end",
        func: function() {
            onClose();
        }
    }

];

var d = new dialog();
d.start(story);
```

```
</script>
```

Listing 2: Code for an Example Dialogue

Each object in the story list contains one or more attributes describing part of the dialogue. The first object in the story will show up in the dialogue screen first, and from there the sequence will follow the labels defined in the rest of the story.

The following is the actual dialogue screen that is produced when the code is run. At the start, the first question will display and the user is given two options to choose from.

Figure 5.1: Initial Dialogue Screen

If the user selects the “yes” option, a popup will briefly appear saying that they should not accept food from strangers.

Figure 5.2: Example of Popup When Wrong Choice is Selected

By clicking on the “no” option, you will be taken to the next dialogue screen. In this case, Selena will simply say she’ll eat by herself and the conversation ends.

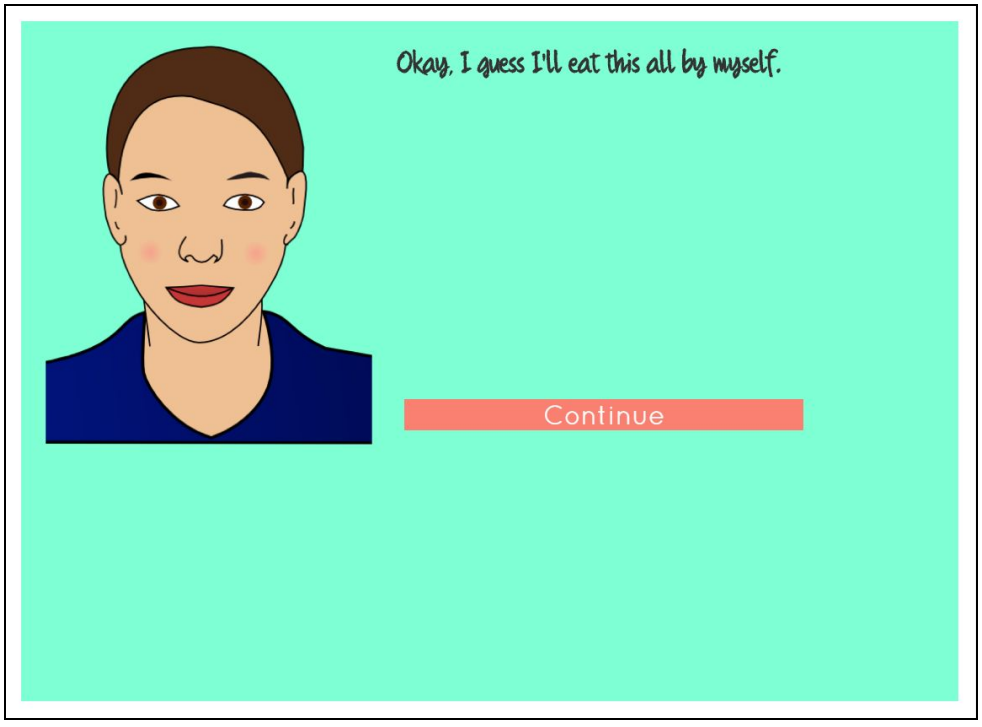

Figure 5.3: Resulting Screen When Correct Choice is Selected

As shown in this example, an object with the `question` attribute represents a portion of the dialogue with a question or text from the speaker, and one or more options for the user to select in response. An object with the `p` attribute represents a popup that simply shows a message to the user, used to indicate that they should choose a different option.

The following is a list of each attribute that is supported and what it accomplishes:

- `label`: a string used to link questions or popups together.
- `question`: the text that is displayed at the top of the screen. May include HTML formatting.
- `responses`: a list of response objects. Each response object contains the `m` attribute for the message text, and the `next` attribute which represents the label to go to when this option has been selected.
- `img`: an optional image that represents the person who is speaking.
- `p`: a simple popup message
- `func`: another optional attribute. This is a function that will be executed when this label is reached. It is primarily used for code handling the end behavior of a dialogue interaction.

This design supports dialogue conversations of considerable complexity. The label system allows for easy implementation of complex dialogue trees. This engine made adding new dialogues to our game a very manageable and straightforward task.

4.5 Click-and-Find Mini-game

The HTML and CSS files necessary for the click-and-find mini-game were mostly hard coded. This was primarily because of the need for the items within the image to be in specific places. Due to the fact that these files are hard coded, neither one is able to be reused for other scenes. However, they can serve as helpful templates that can be utilized in the creation in other scenes. The JavaScript associated with the click-and-find mini-game can be reused for other scenes, only needing the addition of more cases, should that be desired.

In order to accomplish the task of creating a click-and-find game, image maps and images placed on top of other images were used. These two objects were necessary in order to keep the number of image maps and total number of images needed as low as possible. When an image map is clicked, a function in the JavaScript is called in order to replace the main image in the scene with a new image where the item within the image map is now highlighted, making it obvious to the user that the item has been selected. This function also sets up a new image map that removes the portion of the map that was around the item that has already been selected. Smaller images placed on top of a larger image were used in order to put red herrings within the click-and-find game. When a smaller image is clicked, a function in the JavaScript is called that displays a popup message and makes the smaller image hidden.

4.6 Quiz Mini-game

The quiz mini-game was designed to be a simple multiple-choice section that added an educational component to our game. However, in order to maintain the simplicity of the mini-game, the HTML was given a more complex design. **Figure 6** shows the resulting display of the HTML file. This particular HTML file displays the information about Ynes, the quiz, and a results page. The use of CSS styling allowed the game to be displayed in a visually appealing way as well.

Although the HTML for this particular mini-game was a bit more complex, the basic logic in the JavaScript was fairly simple. The main job of the JavaScript was to hide and reveal certain parts of the HTML while also keeping track of the player's input, as shown in **Listing 3**.

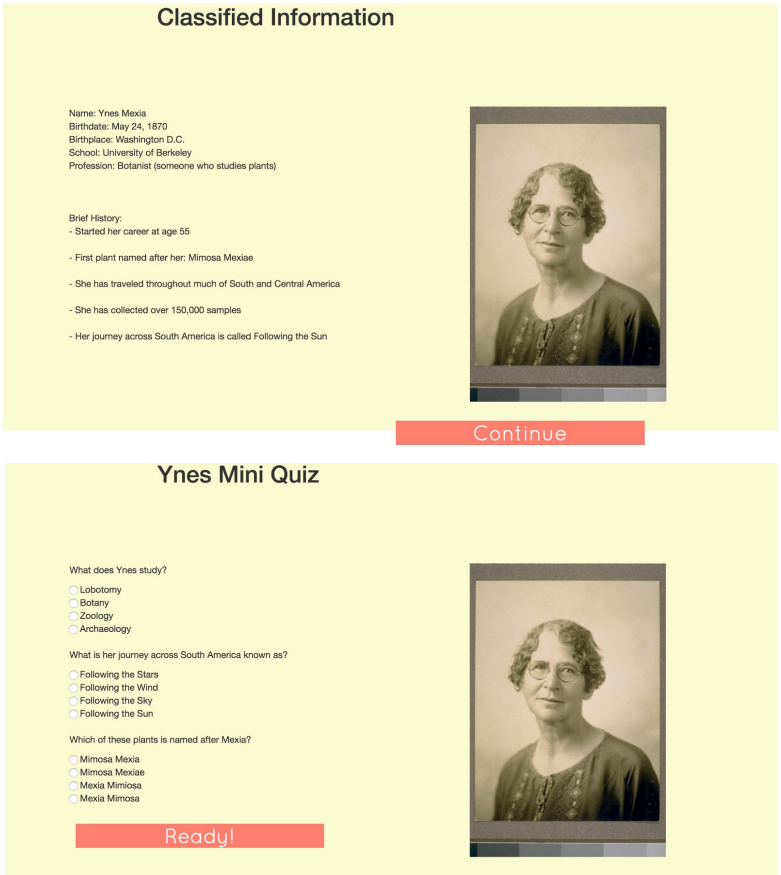

FIgure 6: The three displays of the quiz mini-game

5.0 Decisions

5.1 Implementation Decisions

For the implementation of our game, we decided to use the basic components passed down to us by the previous group. These included:

- The Mapbox interface
- Designing the game using HTML and JavaScript
- The general story of Ynes' journey

We also decided to move the game onto the school's Luna server for security and accessibility reasons.

5.2 Design Decisions

Figure 7 shows how we used a sliding panel to display news feed, which updates with useful information as the player progresses through the game.. We also chose to use the sliding panel to display mini-games.

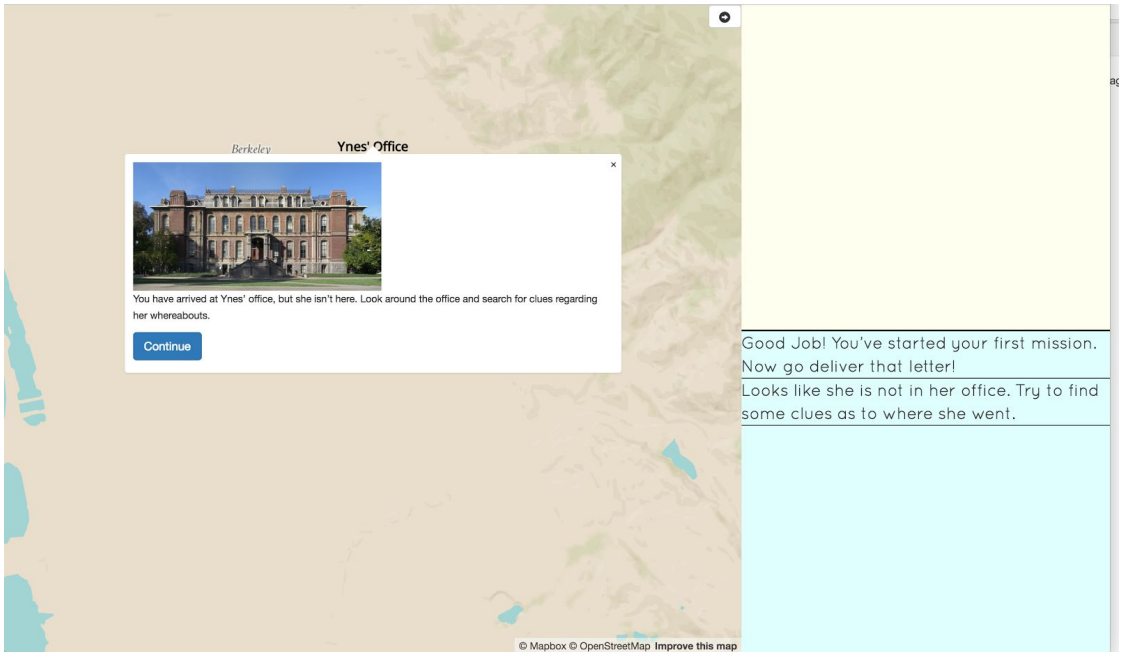

Figure 7: Screenshot of sliding panel containing the news feed

During the game, we chose to disable the player's ability to navigate on the map in order to prevent them from getting lost or wandering off to another point on the map that was not relevant to the game.

The game was meant to be both educational and fun, so the mini-games were divided into two categories: those that were more educational rather than "fun" and those that were more engaging as opposed to educational.

For the mini-games that were actually implemented into the game, we decided on the following:

- Matching mini-game: A game where the player is given a set of pictures and names that they must match together. Animals and plants were the main topic for this games. This mini-game was chosen for its simplicity and ability to be expanded on. It falls under both educational and fun in getting kids to learn about new things.

- Click-and-find mini-game: A game where the player must search within an image for clues by clicking on certain objects. Also used in the cases of multiple dialogues. This mini-game was chosen for its interactive nature allowing the player to actively search and try to find answers.

- Quiz mini-game: A game that has the player answer questions in the form of multiple-choice type quiz. This mini-game was chosen because of the need to ensure that there are educational components within the game. Its purpose is also to make sure the player is paying attention and learns about new things we think are important to know.

- Dialogue mini-game: A game where the player will end up in a conversation with a pre-setup NPC that will respond and ask questions. The purpose is to find the right thing to ask in order to get help to continue on to the next clue. This mini-game was chosen because an aspect of mystery games is the conversations to find clues. While looking at other mystery games, things like interrogation or asking around were important for the story.

We also came up with other ideas for games that we initially designed and planned on incorporating into our game, but were not able to implement due to time constraints. These included:

- Boat Fixing mini-game: A game that required the player to solve a puzzle by putting pieces of a boat back together. This mini-game was chosen so that there would be a fun mini-game that could act as a break between the story and other education-focused mini-game.

- Boat Transport mini-game: A game where the player must transport supplies across the river. This mini-game was chosen since it was more of a complex puzzle-type game that would get the player to think and use problem solving skills. It contrasted well with the other mini-games by being more focused on logic, which expanded on the topics that players could learn.The general idea of the game is:
    - The player has an amount of load on one side of the river
    - The player has a boat that can only move a certain amount of load at a time
    - The boat also has a limited amount of times that it can go across
    - The objective is to move all the load on one side of the river to the other in the limited amount of time

- Boat Maneuvering mini-game: A game where the player must avoid and dodge obstacles as their boat goes down the river. This mini-game was chosen because there was need for another mini-game that was more fun in order to keep players from losing interest. This mini-game also was chosen because Ynes talked a lot about maneuvering through rivers.

6.0 Results

By the end of this field session, we created a high-functioning game that fulfilled our goal of creating an engaging and educational game for children that tells the story of Ynes Mexia's journey. There are a number of aspects that show how much we accomplished over the past five weeks. We were able to integrate the route of the player searching for Ynes into Mapbox via map markers with popup messages, and included a number of mini-games as well. These games include a clue-based click-and-find game, a trivia/mini-quiz game, two games in which the user matches a term with the corresponding picture, as well as a number of dialogue sequences that increase in complexity as the user progresses through the game. In addition to the functional components of our game, we had to create a game that was visually appealing and interactive beyond simply clicking buttons to continue. To accomplish this, we added a number of pictures and background music that engage the user in various ways. Our documentation is also very detailed and easy to follow, which will be critical for any future additions that others may want to make to this game.

Though we did create a game that we believe satisfies the criteria given to us by our client, there are some features that we were not able to implement due to the time constraint of a six week field session. We wanted to add in more complex animations throughout the entire journey in order to make the game even more interactive than it already is. There were also additional mini-games that we designed but were not able to implement. Our designs included both educationally-based games and games that are less educational but intended to be more "fun" for the player. It was important to have a variety of games so that the user would gain some knowledge while at the same time enjoying our game. Some of the more educational games we were planning on implementing included a river-crossing logic puzzle and a boat-fixing game where the user would answer various questions in order to fix each part of the boat. For the more engaging games, we planned on implementing navigation games in which the user would have to avoid obstacles in their path. Unfortunately, our limited time did not allow us to implement every game we wanted to and forced us to have to make decisions on which mini-games were vital to our project and which ones were not as important. Because we documented our project so well, however, these could be additions that others may decide to implement in the future with relative ease.

Another aspect of our initial plan that we were not able to complete was gathering constructive feedback from a group of users. To test our game, we wanted to utilize user-based testing for our game. We planned on collecting a group of approximately 10 to 20 middle school students, let them play our game, and provide feedback to us. For the feedback, we developed two separate surveys: a pre-survey and a post-survey. The pre-survey was designed to give us an idea of what types of topics children enjoyed studying and what characteristics they found to be

most appealing in a game. The post-survey would give the students a chance to provide any comments and/or criticisms of our game. Once we compiled the data from the user-based testing group, we would be able to make any necessary changes to our game. We contacted Dr. Tracy Camp and she told us that she would communicate our request to one of the summer youth camp coordinators and try to gather a group of students. However, we never heard back from the camp coordinator and therefore were not able to secure a group of students for testing. Once again, though, this is a part of this project that can be utilized by Dr. Gianquitto in the future.

In summary, we brought together research of our own and the research of previous groups to build a functioning game with a start and finish. We created a solid foundation on which future groups can rely to add new or modify existing content with relative ease. We hope that this project will only continue to improve as time goes on.