HACKMAIL

COS426 Spring 2020
Final Project Written Report

Abigail Rettew, Jamie Guo,
Jessica Fielding, & Ze-Xin Koh

Abstract

Hackmail is a game where you navigate your professor’s laptop to elicit evidence of his extramarital affair so that you can blackmail him. Solve puzzles on his computer to gather enough pieces of evidence.

Introduction

Goal

What did we try to do?
Our goal was to build a game where you solve different minigames in order to win the overall game. The minigames share the common theme of eliciting evidence against your professor’s extramarital affair. Some minigames require some interaction with the 3D scene to gather information.

Our Minimum Viable Product (MVP) was a simple 3D scene, with at least one minigame. Stretch goals included the inclusion of more games.

Who would benefit?
Anyone who enjoys games!

Previous Work

What related work have other people done?
Hackmail is loosely inspired by the popular "Keep Talking and Nobody Explodes", a local co-operative game where you solve many minitasks on a bomb before a timer runs out and the bomb explodes. We decided it would be exciting to try a similar many games within a game approach.

The text bubble game is inspired by a popular phone app called Fruit Ninja. The slider game is inspired by the COS 226 Slider puzzle assignment.

Approach

What approach did we try?
Style: We decided to aim for a more light-hearted, stylised aesthetic, due to the time constraints. This informed our decision to base the desktop and dialog boxes (for game wins and game over screens) off of the classic windows wallpaper and dialog boxes.

Methodology

What pieces had to be implemented to execute my approach?
One design challenge that we encountered was in deciding how to navigate the switch between the 2D and 3D components of our game.

Toggling between 2D and 3D

Our starting scene consists of the 3D room where the professor's laptop sits on a desk. Clicking on the laptop opens up the 2D desktop and the minigames it contains. Clicking "Esc" when you are on the desktop allows you to return to the 3D room.

The final project seed provided starter code for building a 3D web application using ThreeJS. We implemented the 3D portion of the game using this starter code as it was a natural and recommended solution; however, the implementation of the 2D portion was not as clear-cut. We considered multiple approaches for rendering 2D graphics and toggling between 2D and 3D.

1. The first possible approach we considered was to use ThreeJS scenes and objects with a static camera to simulate a 2D screen. The main advantage of this approach was that it worked well with the project seed code by keeping everything within ThreeJS. Displaying objects, changing scenes, fixing a static camera, and disabling camera controls would all be straightforward. Despite this advantage, using ThreeJS seemed like a wasteful and unnatural solution to the problem. Rendering 3D objects and scenes just to simulate a 2D screen would be significantly more resource-heavy than necessary. Significant extra work would also need to be put into tasks such as tracking pointer placement through ray casting and manipulating mesh placements to appear two-dimensional.

2. The second approach we considered was to use pure JavaScript with HTML elements and CSS styling to render the screen and create the 2D minigames. While this would probably lend itself well to creating the desktop, which is a 2D screen with image buttons linking to each minigame, it didn't seem to be an elegant way to implement the 2D minigames, which would require many moving, dynamic, and interactive parts.

3. The final approach, and the one that we ultimately decided on, was to find and use a 2D graphics library and combine it with ThreeJS to build our game. The main drawback of this approach was that it would require us to learn a new library and figure out how to smoothly integrate it into the existing seed code. However, this was the most natural solution, and a 2D graphics library would allow us to focus on building fun minigames without spending a significant amount of time on lower-level rendering techniques.

Our implementation of this third approach involved creating two separate WebGL apps, one 3D app built using a 3D renderer and one 2D app built using a 2D renderer, and defining overall wrapper functions for event listeners and frame updates to call the relevant functions from the correct app based on game state.

Implementation

We used HTML/CSS/JQuery for the GUI overlays, the ThreeJS library for the 3D scene, and the PixiJS library for the 2D desktop and minigames. Here, we describe the various components of our game:

Landing screen
The landing screen has the game's title Hackmail, and the names of our team members. Clicking on the Play button leads you to the "instructions" dialog box overlayed on the room scene.

Instructions
The instructions dialog box contains basic instructions for the game. It is displayed right after the landing screen, and can be revisited at any time upon hitting the key "P".

Room
The room consists of a desk with a laptop on it, as well as a notepad with passwords that are clues for the Hangman game. Clicking on the laptop causes the desktop to render.

Desktop
The desktop contains icons that link to each minigame. Clicking the escape key at the desktop or within any of the minigames returns the player to the 3D room scene.

Games
Each minigame extends a general game class, which contains general functionality, such as the navigation bar, the minigame win and game over mechanisms.

Minigames

Text Bubbles

A bubble is generated with some probability for every frame. Each bubble starts at the bottom of the screen and is given a random x and y velocity. Each bubble contains either a scandalous or innocent text with random probability. Clicking on a scandalous text causes your point score to increase, while clicking on an innocent text causes your number of lives to decrease.

If your number of lives drops to 0, it's game over for the minigame. If your score reaches the threshold of 10, you win the game.

It was initially challenging to figure out how to remove bubbles that have fallen off the screen, to prevent crashing the browser with too many objects. We ended up checking for bubbles that have fallen below a certain y-position, and filtering them out of the set of bubbles, to prevent them from updating.

Slider Puzzle

A basic slider puzzle! This was fairly straightforward to envision and implement. Each block is a textured sprite. Upon clicking on one of the boxes, the game determines if the block is next to the empty spot and, if it is, slides the block into the empty position. After every move, the game then checks if the pieces are ordered correctly.

The most noteworthy part of this game was probably the animation, which we added after the core of the game was complete. The animations themselves are pretty simple (a fade-in for the final piece once the puzzle is complete and the pieces themselves sliding into their new positions). We would have liked to use a Tween library and callbacks if available, but we eventually decided to use a simpler solution using incrementing and conditionals.

Hangman

Hangman features three different types of letter blocks. The first (and first implemented) were the password blocks which act as the "password" you're trying to crack. The second are the random blocks that pop up and allow you to guess letters. The third type are the "wrong letter" blocks, which display incorrectly picked letters.

There was some consideration over gameplay elements for this game. We tinkered with some parameters a bit to decide how random the random blocks should really be. Currently, we choose with a fairly high probability to spawn a letter that's definitely in the password, though we also played around with higher/lower probabilities and with spawning specifically unrevealed letters in the password. We also tried a few different methods of dealing with incorrect letters (should you be penalized for choosing the same incorrect letter? How many tries did you get) before settling on the current system of allowing the same number of unique incorrect choices as in the password itself.

Finally, this game is tied into the "real" 3D world, which provides the list of possible passwords. This took a fair bit of experimentation; the page moved from being initially fixed in place to eventually being responsive to the camera position.

Results

How did we measure success?

We asked friends and family to navigate the game and provide us with feedback throughout the production process. Overall, people thought the idea was fun and enjoyed it.

What experiments did we execute?

Based on feedback from people we've asked to try our game, we adjusted certain aspects of the game. For example, we tuned the difficulty of the minigames, such as slowing down the speed of the text bubbles in the text bubble game after feedback that it was too fast for the text to be read.

Discussion

Overall, is the approach we took promising?

We believe that the approach we took in structuring our project and toggling between 2D and 3D scenes was promising.

What different approach or variant of this approach is better?

We believe that the approach we took to toggling between the 2D and 3D components of our game was the most natural.

What follow-up work should be done next?

Given more time, a promising direction would be to include more interactions between the 2D and 3D scene, such as our stretch goal of including periodic interruptions by the professor into the room, where the player would have to hit a key to hide under the desk, regardless of their progress through the minigames.

It would also be great to add more minigames, or some way of randomly generating varying difficulties of minigames according to the player's progress through the overall game.

Another consideration was adding a timer that counts down throughout the game, and if the player fails to complete some number of minigames before the timer ends, the game is over. This would add to the suspense of the game.

What did we learn by doing this project?

We learned a lot about the way a browser window updates and renders scenes, and how asynchronous events are handled. We also definitely developed a deeper understanding of rendering three dimensional scenes using the Three.js library, and two dimensional scenes with the PixiJS library. As with any more open-ended project, we also learned a lot about time management and managing the structure and development flow of a project with many team members.

Conclusion

How effectively did we attain our goal?

We managed to build our Minimum Viable Product of a simple 3D scene, with at least one minigame, and expanded that to include two other games.

What are issues we need to revisit?

We would like to revisit the overall coherency of our game, so that it doesn't feel just like separate minigames. This could be done by making more minigames that have components of interaction with the 3D scene, where the player might have to poke and prod around the desk or room to discover an important clue.

Contributions

Ze-Xin	Text Bubbles minigame, minigame win and end game mechanisms, some general game class functionality
Jamie	Toggling between 2D and 3D, landing page and instructions, general game class, overall game winning mechanism
Abby	Hangman and slider minigames, 3D scene
Jessie	Audio, graphics, 3D scene additions

Works Cited

1. https://nodejs.org/en/
2. https://webpack.js.org/
3. https://jquery.com/
4. https://threejs.org/
5. https://www.pixijs.com/
6. https://keeptalkinggame.com/
7. https://fruitninja.com/
8. http://www.artbylogic.com/puzzles/numSlider/numberShuffle.htm?rows=3&cols=3&sqr=1
9. https://glb-packer.glitch.me/