Mazeophobia Intermediary Report

COS426 Final Project

Watson Jia & Sten Sjöberg

Abstract

Mazeophobia is a first-person horror game developed using the ThreeJS library. The game is set within a randomly generated maze, and the goal of the game is for the player to escape the maze by finding a finish beacon located at the opposite end from the player spawn point. Artistic emphasis is placed on the horror element, and as such the maze is set within a dark environment and the player only has a flashlight to navigate through the maze. We aim to finalize core gameplay features such as game loss mechanics and continue to enhance the horror aesthetic through visual effects and textures in addition to adding sound effects and musical ambience.

Introduction

Goal

The goal of this project was to implement a web-based horror game with a simple premise and relatively uncomplicated gameplay mechanics. Given our limited timeline, we aimed to create a simple baseline game but could be expanded upon as necessary to deliver a more polished product. We also aimed to have our game contain some degree of replayability as well as have a good emphasis on the horror aesthetic and mechanics.

Previous Work

Our project was largely inspired by existing horror games such as Amnesia and Phasmophobia which manage to achieve much in terms of horror aesthetics through visual effects and ambience. The setting of our game was inspired by Pac-Man, in which the eponymous character navigates through a maze while trying to avoid enemies. We wanted to combine gameplay elements from all three games to create a new simple horror game that is straightforward yet enjoyable.

Recently, there have been a few COS426 final projects designed as first-person games, but to our knowledge, there has not been a horror game proposed as a final project since Blenderman in Spring 2019, which is a first-person horror game inspired by Slenderman. Our goal was not to build on top of Blenderman but rather go in our own unique direction and attempt to offer more in terms of horror aesthetics.

Approach

We build our project in ThreeJS, as it is a well-documented framework for developing WebGL applications. Moreover, our prior experience using ThreeJS for the COS426 assignments made it an attractive choice for our final project. Our runtime environment was chosen to be Node.js which is a very popular server-side Javascript framework capable of supporting high-performance web applications.

Our approach to designing and building this game was inspired by the approach taken by Phasmophobia, which has a simple premise and easy-to-grasp gameplay mechanics but achieves a convincing horror element through its ambience and visual aesthetic. Given that our final project is heavily limited by time, we wanted to be able to quickly build a basic game that could be expanded on as necessary to refine and polish the product. While we don't have the resources to deliver an expansive game as Amnesia, we believe that there are a few basic gameplay mechanics and visual and audio features that can go a long way to delivering a horror element.

Methodology

Our contributions to the core gameplay mechanics so far can be grouped into four main categories: maze generation, flashlight implementation, first-person controls, and scene transitions.

Maze Generation

A randomly-generated maze is the main environment in which our game will take place in. The maze generation design is split between two components. The first is a graph-based data structure which defines a maze. The second is the code which creates the graphics representation of the graph maze data structure.

The data structure first defines a grid of cells with dimensions **n** x **n** for a given width **n**. Then, between each cell, an edge is defined which should be thought of as a wall. At this stage, each cell is entirely disconnected from the rest and the fundamental data structure is set up.

For the maze itself we wanted some few properties. First, we wanted a perfect maze, meaning the whole maze can be defined as a tree. All cells are connected in such a maze with exactly one path between any two cells. Further, we wanted a relatively simple maze, without dead ends which were too long, potentially causing frustration with the player. We also wanted a degree of replayability, and having only one maze would not provide any replayability once the player solves the maze for the first time. Finally, we wanted a generation algorithm which was relatively simple to implement and with a not-unreasonable runtime.

A randomized version of Kruskal's algorithm served our purposes well. The original Kruskal's algorithm finds a minimum spanning forest for an undirected graph with weighted edges. The algorithm is defined as follows:

1. Assign a unique group number (**gid)** to each vertex.
2. Pick the lowest-weight edge in the structure. Only if the edge runs between two vertices with different **gid**, remove the edge and combine the two groups under one of the vertices' **gid**, thus removing one group.
3. Perform step **2** until the number of groups in the structure is exactly one.

The randomized version is identical save that in step 2, no edge-weights are considered, and an edge is simply chosen at random. This, rather intuitively, can be used for maze generation. Vertices are considered as cells, groups as continuous rooms, and edges as wells. Rather intuitively, this algorithm results in a set of remaining edges which separate the cells such that a random perfect maze of one room is defined.

We considered several other algorithms such as randomized Prim's algorithm and Wilson's algorithm. Though these algorithms have their own advantages (mainly a different generation bias and the lack of generation bias, respectively), we were happy with the results from Kruskal's.

In our maze data-structure we also defined an internal 2D cartesian coordinate system identical to a would-be image where each cell corresponds to a pixel, with which we defined the locations of each edge in the graph.

The second step, to instantiate the defined maze for the game, is very straight-forward and consists of untextured **BoxGeometry** objects, rendered as Phong materials. Simply, for each remaining edge in the maze data structure, a wall is spawned at the edge's location, based on the **cellWidth** length constant. The orientation of the edge was also considered, and the wall was rotated accordingly. With each wall at a width of exactly 9/8 of a cell width, the walls are flush with each other in intersections. After the internal maze walls are generated, four perimeter walls are generated, enclosing the maze cell grid.

Flashlight Implementation

With a theme centered around horror, we wanted to give the player limited and directed light, with a marked emphasis on exploration. For these reasons, we opted to create a light source emulating a flashlight held by the player.

This effect was accomplished by strategically placing, and updating, a **SpotLight** object at the camera. Simply, a **SpotLight** with a somewhat constrained luminosity and light angle is added to the scene. In addition, the light's target object and the camera are also added to the scene. In order to position and orient the light correctly, the update function is slightly involved. First, we get the camera's direction vector **dir**, which we normalize. We then define the light position as that of the camera's minus **dir**. We then

define the light's target position at the camera location plus 3 x **dir**. This results in the flashlight being placed directly behind the player, always pointing directly forward. Since the light is just behind the camera, the light shines even when the player brushes against a wall, though in a more constrained circle due to the angle of the spotlight. This ensures that the player can keep track of their surroundings.

First Person Controls

Our first-person controls are implemented such that one uses the WASD keys in order to move the player within the scene and uses the mouse to move the camera around in the scene. The player movement also needed to correspond to the direction of the player camera. We adapted a ThreeJS example which uses the **PointerLockAPI** in order to achieve first-person gameplay mechanics.

Unlike the ThreeJS example, for our gameplay mechanics, it is important that the player cannot clip through the maze walls. To achieve collision detection, each wall within the maze is associated with a **Box3** object that describes the bounding box of the wall. For simplicity, we designed the maze such that walls and their bounding boxes will be axis-aligned. A collision occurs if a player movement causes the camera position associated with the player (represented as a point in 3D space) to be contained within the bounding box of a wall. When such a collision is detected, the camera position is simply set to the nearest position just outside of the bounding box of the wall. To prevent clipping of the first-person camera with the wall model, the dimensions of the bounding box associated with the wall is slightly larger than the dimensions of the wall.

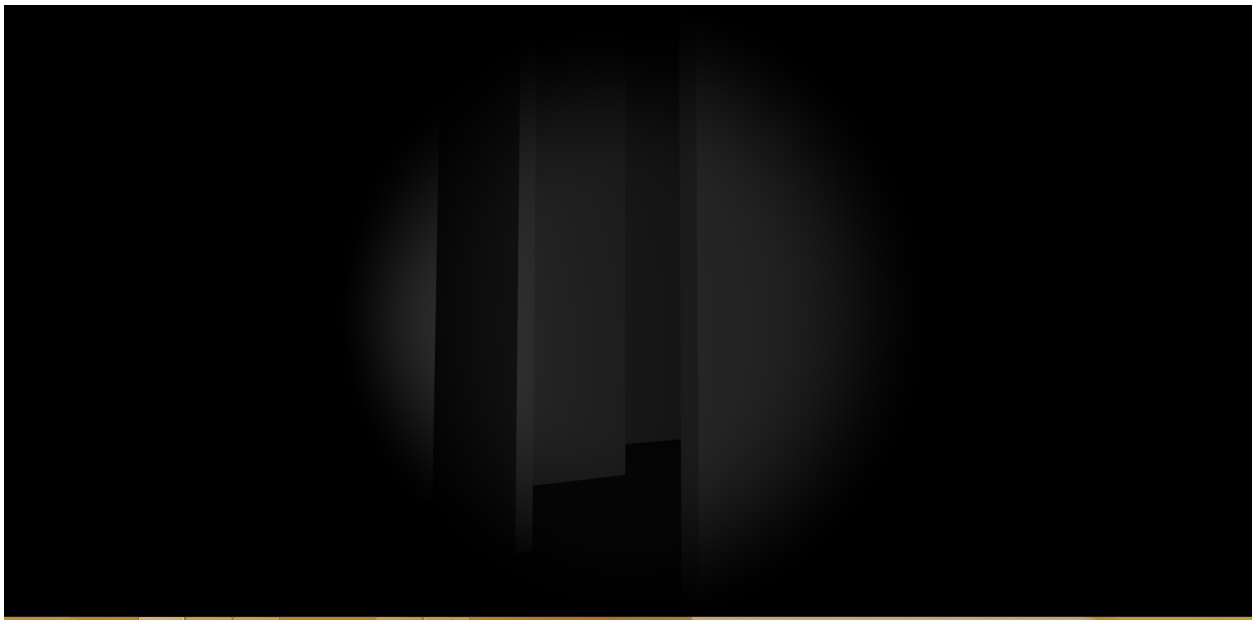
Scenes & Transitions

When a user loads the game for the first time, the user will be greeted by a title screen containing the title of the game as well as a button that will start the game. When the player clicks the button to start the game, the user will immediately find themselves viewing the first-person camera of the player inside the maze. This functionality was achieved by having the app.js file render the desired scene and transition between scenes as necessary. There are currently two scenes: **StartScene** and **GameScene**. **StartScene** displays the title of the game as well as a button to start the game. The button contains an event listener that, when clicked, disposes of **StartScene** and then creates and renders **GameScene**. **GameScene** contains the core gameplay environment, namely the player and the maze itself.

Results

With the maze generation algorithm, flashlight implementation, and first-person controls fully implemented as well as most of the scene transitions, we currently have a working rudimentary prototype for a game. A birds-eye view of the maze shows that we have a viable random maze-generation algorithm and our first-person controls within the maze show that we have implemented the most critical gameplay mechanics. Moreover, we have some basic horror ambience that will be expanded upon.

A birds-eye view of the maze



A first-person view inside of the maze

Discussion

Overall, our final project appears to be coming together quite nicely, with most of the core gameplay mechanics and a few basic horror elements implemented already. We still have some more work to do in order to have a more polished and finished product for final presentations.

Next Steps

We aim to finalize win and loss mechanics in order to finish our minimum viable game. A player will win the game when the player locates and touches the finish beacon at the end of the maze, while a player will lose the game if the player takes too long to find the finish beacon. We aim to implement this feature as “player sanity” which will steadily decrease over time, and the lower the sanity, the more horror elements will be introduced. These win and loss conditions will also be associated with victory or loss screens. Once this is done, we aim to enhance the visual and audio aesthetic in order to have a more convincing horror game. This will involve including textures on the maze walls and floor as well as audio mechanics such as footsteps or heartbeats to improve on the existing horror ambience.

Conclusion

We were able to implement a rudimentary first-person horror game using the ThreeJS framework. We have a working alpha version of the game that we see being a full minimum viable product within the next couple of days, and from there we will be able to focus fully on building upon the horror mechanics and ambience. So far, we have learned a lot about web-based game development in JavaScript through this experience, and we are glad to have more time to be able to continue working on this project and present a product that we are excited about during final project presentations.

Contributions

Sten implemented the maze generation algorithm that randomly generates the maze in which our game takes place. Sten also worked on placing the walls of the maze that correspond to the maze generated by our algorithm as well as the player flashlight mechanics. Watson worked on implementing the player controls that allow for first-person camera movement with the mouse and player movement with the keyboard. Watson also implemented collision detection with the maze and handled the game transitions between the start screen, gameplay, and end screens.

Works Cited

https://en.wikipedia.org/wiki/Amnesia:_The_Dark_Descent
https://en.wikipedia.org/wiki/Phasmophobia_(video_game)
https://en.wikipedia.org/wiki/Pac-Man
https://github.com/gnuoyohes/blenderman
https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/PointerLockControls.js
http://personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.htm