CMPT 370 AS01
Final Project Report

Stefan - Alec - Juan

Fall 2024

Table of Contents

What Our Game is:

Our game is a first-person FOV maze game inspired by Doom (1994) and Maze War (1974). We initially wanted a shooter-style gameplay as it seemed like a straightforward implementation, but we soon realized that would not be possible with our limited time constraints. The final product is more of a tech demo showcasing what we learned in the course; ray tracing, linear interpolation, etc. Aside from using what we learned in class, we also looked into other techniques with the aid of tutorials during development.

In our game, the player is able to move around and roam the map. There are stairs that will change the player's perspective on the y-axis and walls that will prevent the player from walking through the map. Unfortunately, we were not able to implement textures and lighting as we had a significant amount of assignments from other courses; instead, we focused on having a functional 3D presentation of the world with objects and collisions existing in the rendered map.

Games Like Ours:

We wanted to keep the style of Doom but shift the playstyle to a maze-type game, similar to Maze War.Here are a few other examples of similar games like ours:

- Wolfenstein 3D (1992)
- Duke Nukem (1996)
- Gloom (1995)
- Faceball 2000 (1991)
- The Quake series (1996)
- Project Warlock (2016)
- Selaco (2024)

What We Did:

In our project, we created a python based 3D game similar to Doom and Maze war. This program heavily relies on the pygame library to calculate vector math and other aspects such as drawing lines. Our game implements the fundamental components of a 3D game such as Binary Space Partitioning for scene rendering, real time player movement in the first person point of view, and map traversal in a 3D format.

WAD FILE

All of the information for our map and the layout of those segments, lines, nodes for our bsp tree, pretty much everything, all came from the DOOM wad file. Here we parsed the file to get the map format for the original DOOM game and stored data like vertices, linedefs, sectors, and subsectors. The WADReader.py script holds the main functions for the purpose of parsing the WAD file.

Map Rendering

After the WAD data was correctly collected, we then created the 3D map structure using MAPRender.py. In this script, we re-map game coordinates to our screen space and render elements of ou maps, like sectors, linedefs, and walls. In the initial state of creation for our game, we debugged using a 2D map view of the DOOM map. Debugging players FOV, vector span, and bsp was all tested in the map rendering program, prior to 3D implementation.

3D Visualization and BSP

To visualize and optimize the map render space, we used Binary Space Partitioning (BSP) - although not done traditionally as we have the nodes and leaf nodes of our BSP tree already located in our WAD file. We just recursively parsed through those nodes already set and traversed our BSP tree with respect to the player. This was done to have an efficient map render and render the aspects we wanted only. This took in the player's relative position with respect to the screen and also took the root node of the BSP tree when traversing the tree to find the most efficient render.

Player

We created a simple first-person FOV to simulate a character. The player camera is able to move the player's position, pan the camera, and detect collision as well. Movement is done with a fixed speed and rotational speed set in Settings.py. The player height is dynamically calculated based on the subsector of the floor heights to simulate elevation changes.

3D Wall Rendering

Wall rendering was done with SegmentHandler.py, which calculated the wall sizes and heights based on the player's position and perspective. We also implemented a solid wall and portal wall function to handle the creation of portal walls and normal walls.

Main Game Loop

The main game loop runs in main.py. Here the game maintains 60fps on a 720p resolution (Hoping as that is what we hardcoded to display). The fps is also shown in the top left of the screen. This was done for debugging purposes.

What We Learned:

Our project was a great way to learn and implement new concepts. One of the most interesting aspects to implement was the BSP tree. Seeing how the map was generated and watching the tree traverse in real time was both engaging and rewarding. Another highlight was transforming the 2D map into a 3D-rendered world. Initially, we worked with basic lines, but as we calculated wall dimensions and incorporated the camera's perspective, it was exhilarating to see fully-formed walls materialize. This progression brought a sense of accomplishment as the game environment began to take shape. We also gained a deeper appreciation for vector mathematics. A big portion of the player, collision, and wall generation, is all based on vector math. While vector math we already knew, we felt we gained a better understanding and appreciation of the fundamentals of displaying a camera and having a first person FOV. Learning perspective, scaling, and screen transformation aided us in being able to understand the importance of transferring points from a space to an actual screen. Additionally, building a functioning game and using pygame as a whole was another huge learning experience. We have never used pygame so to utilize the functions and see the capabilities of pygame was a learning experience. It truly put us into the shoes of early developers and how they would create a game like this. Finally, parsing WAD files and debugging the system was a significant challenge that reinforced the importance of patience and precision. Tackling binary data parsing initially seemed daunting, but overcoming this hurdle gave us a newfound respect for the intricacies of working with binary formats and the foundational structures that power games like this

Individual Contributions:

Alec:

- Implemented 3D rendering of map, including portal walls, portals and render od line defs and FOV
- Implemented player height control to dynamically change based on lower sector of walls
- Implemented Collision for player to interact with walls
- Player camera and movement in 3D world

Stefan:

- Parsed WAD data to enable its integration with pygame, collaborating closely with Alec to implement a BSP tree for efficient world rendering.
- Implemented the rendering of the 2D map, which allowed us to easily confirm whether or not the BSP tree was functioning as intended
- Testing and debugging code to ensure proper functionality

Juan:

- Initialized and maintained the Github repository and pull requests
- Worked on player class and movement
- Testing and debugging
- Aided with troubleshooting code