Doom (1993) using Pygame

Final Project Report

Compiled by:

Brian Yuktipada (2802523004)
L1BC

Algorithm and Programming
BINUS University International
Jakarta
2024

TABLE OF CONTENTS

Chapter 1

PROJECT SPECIFICATIONS

1.1. Project Description

For my final project of the first semester, I decided to recreate Doom (1993), created by id Software, in a Wolfenstein-style game using Python. Originally, I had other concepts before the idea of Doom was finalised but unfortunately, these concepts aren't compatible with Python, especially PyGame. While making Doom, I decided to use the original assets for most of the game's assets while having a few original assets solely for textures.

The goal of this game is to eliminate all the enemies in the game in order to achieve victory. The gameplay for this game is very simple. The player has a single weapon that they can use to defeat the enemies in the game while avoiding taking damage from them to prevent the game ending in a defeat. The player can move around the area to find enemies and defeat them quickly before they get killed. There are about 21 enemies in the game that the player has to eliminate to clear the game, where there are 3 different types of enemies in total. Should the player finish the game in either a victory or defeat, the game will have that victory or death screen for a few seconds before restarting the game to the beginning.

1.2. Project Link

The GitHub Repository of my final project can be accessed through the link provided below:

My GitHub Repository

1.3. Essential Algorithm

1. Enemy AI:

The enemies have several important variables in deciding their actions:

A. Idle Action

The enemies start with being in an idle animation the moment they are spawned. This animation only happens if the player is out of their field of vision. This animation consists of 8 images of the enemy facing different directions (north, northeast, east, etc). The idle animation consists of the 8 images being played in a loop, making it look like they are turning around to scan for the player.

B. Movement

Once the enemies see the player in their field of vision, depending on the distance between the player and enemies as well as the enemies' attacking distance, the enemies will move towards the player and attack them. The enemies' walking animation consists of 4 images of the enemy moving slightly to the left and right, looping them quickly to make it feel like they're actually walking. If the player is spotted by the enemies while being close enough to them, the enemies will attack instead of walking towards the player.

C. Enemy Pathfinding

When there's an obstacle between the player and the enemies in a grid-based game, the enemies can use a pathfinding algorithm to navigate around the obstacles and reach the player. In this case, the Breadth-First Search (BFS) algorithm is used. Given that the game map is composed of grid tiles, BFS is an efficient choice for finding the shortest path. The BFS algorithm will use the grid map to determine the quickest route for the enemies to reach the player.

D. Death Animation

Each enemy has their own amount of health points depending on the type of enemy. When their health points get reduced to zero, it will trigger a death animation that consists of a number of images played in order to make it look as if they're dead or in this case, their bodies suddenly explode into a fleshy pile.

E. Attack Animation

If the player is within the range of the enemies' attacking distance, the enemies will trigger an attack animation while dealing damage to the player. While attacking, the enemies will remain still, so it's easier for the player to aim at them though still at risk of being harmed by the enemies. The attack animation usually consists of two images of the enemy firing and not firing but still aiming at the player played in a loop until the player is out of the range of their attacking distance.

F. Pain Animation

If the enemies were to take damage by the player, a pain animation that consists of a single image of them in pain will be triggered whenever they are hit by the player.

2. Ray casting Algorithm:

Since I made the map of the game using a 2D grid map, I used a ray casting algorithm to turn the 2D Map into a 3D map in order to create a field view of the game world from the player's point of view. The reason I used ray casting is because ray casting is efficient for rendering 3D views,

especially for games where the environment is represented as a 2D grid map. It calculates the visible part of the environment by tracing rays from the player's viewpoint, reducing the number of calculations compared to full 3D rendering. This will also prevent the game from being laggy, increasing the performance of the game.

3. Keypress Algorithm:

Using PyGame's event.get() function to scan whenever the player presses a key in order to process certain actions in the game. This also includes the player's mouse buttons.

4. Event Manager Algorithm:

This algorithm helps check for the current state of the game. The only events in the game are if the player beats or fails the game. Once either of these events have finished, the game will restart automatically, moving the player to the starting area of the game. If the player wants to leave the game, they only need to press the "esc" button, which will exit the program.

5. Enemy Spawn Algorithm::

The moment the game starts, the game will spawn 20 enemies in random positions on the map. I made it so they don't spawn in the obstacles otherwise it would be impossible to clear the game. I gave each type of enemy a probability for spawning. In my case, one enemy has a 70% chance of being spawned, and another with a 20% of spawning. Meanwhile, I had the boss enemy of the game manually spawned at the final area of the game since I still don't know the exact way to make it so the boss enemy would only spawn in the final area and it might risk spawning in the early areas.

1.4. Modules

1. PyGame:

The PyGame is what made most of this game functional in the first place. Without it, it would be nothing. Most of the files used for the game relied on the PyGame module a lot. It allowed me to use the mixer to play audio such as the background music, the sounds of shots fired from both the player and enemies as well as the pain and death for both the player and the enemies.

2. Sys Module:

The sys module was only used to close the game with the sys.exit() method. It has no other purpose other than that.

3. OS Module:

The OS module was mainly used for the sprites such as the player's gun and the enemies' animations. It helped with the filepath of the many sprite images I had in my sprites file.

4. Random Module:

The Random module was only used to help with spawning the enemies in random locations on the map. By using the x-axis and y-axis of the map, it randomly chooses coordinates of where the enemies will spawn as long as it's an empty space.

5. Math Module:

The Math module was the top second most used module for this game. This module is mainly used for the statistics of the game such as the player and enemies' statistics like health and damage as well as assisting with ray casting, and more. It also helps with the game's settings such as the resolution, frames per second (fps), sensitivity and more.

6. Collections Module:

The Collection module is used for managing game events, tasks, or actions made by the player and enemies. It'a also used for backtracking algorithms and maintaining a fixed-size window of the sprites and events.

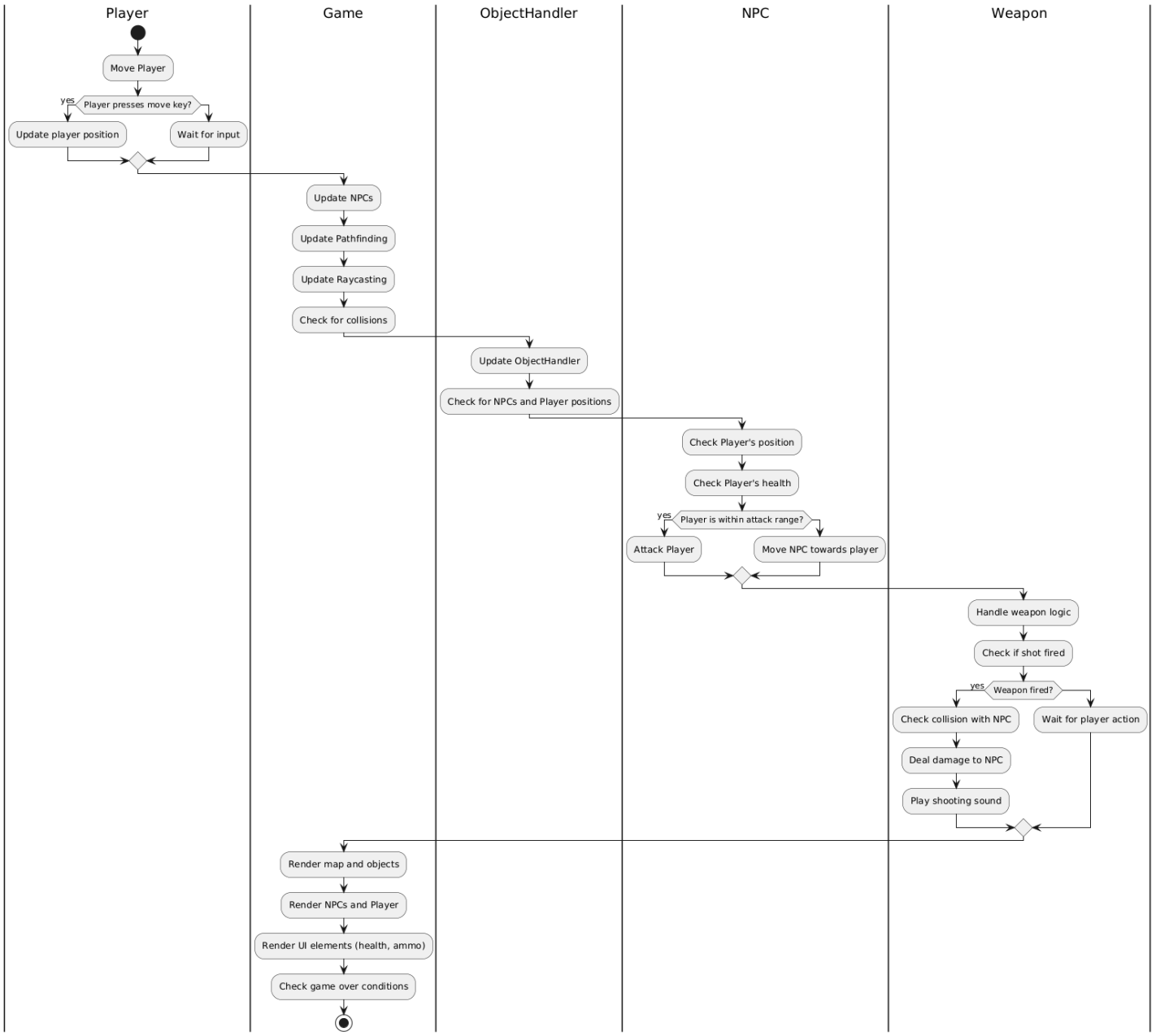
7. Functools Module:

The lru_cache decorator from the functools module is used to cache the results of the pathfinding algorithm. This helps in optimizing performance by storing the results of previously computed paths, so if the same enemies and player locations are queried again, the cached result can be returned quickly without recomputing the path.

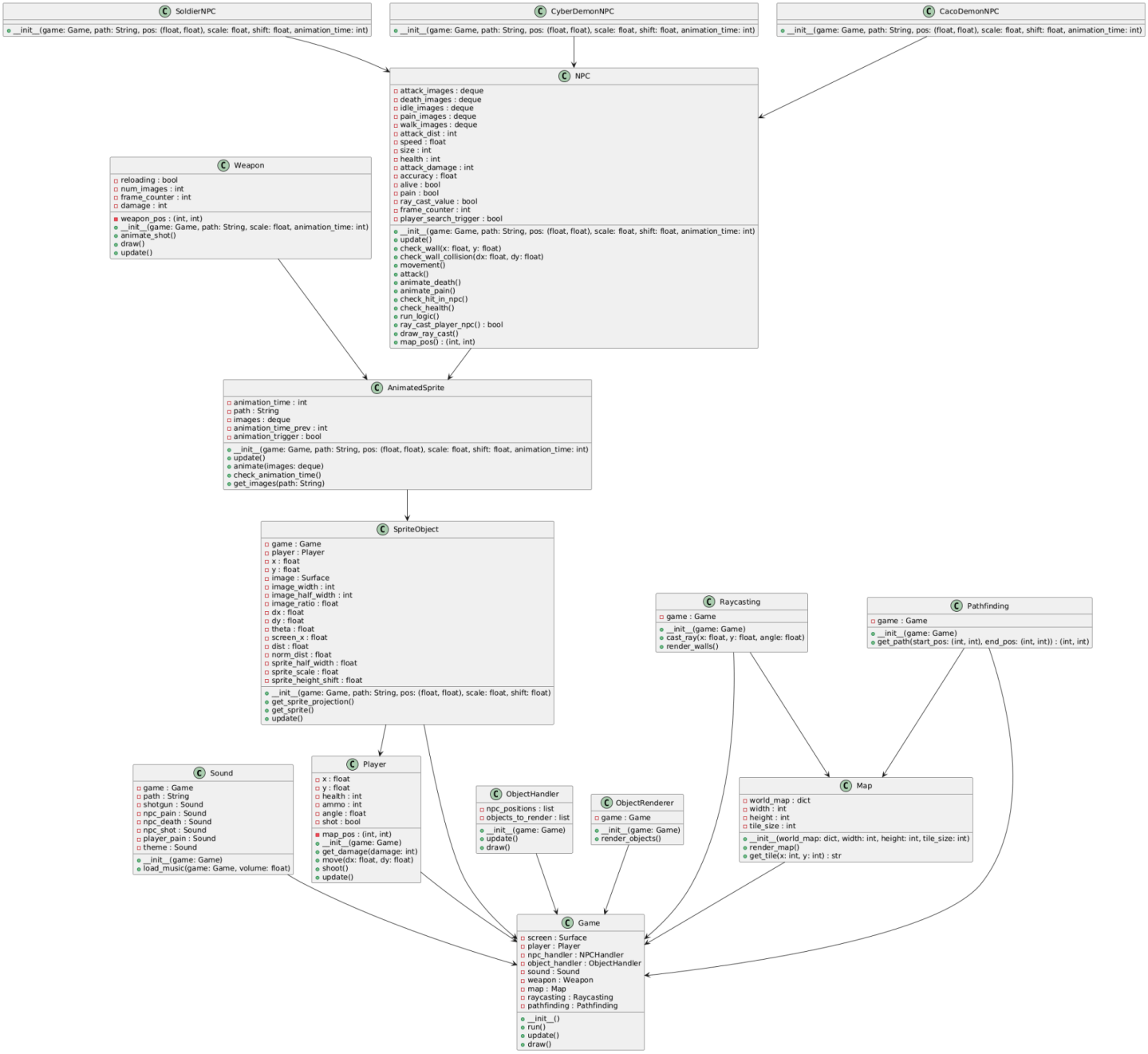
Chapter 2

DIAGRAMS

2.1. Activity Diagram



2.2. Class Diagram

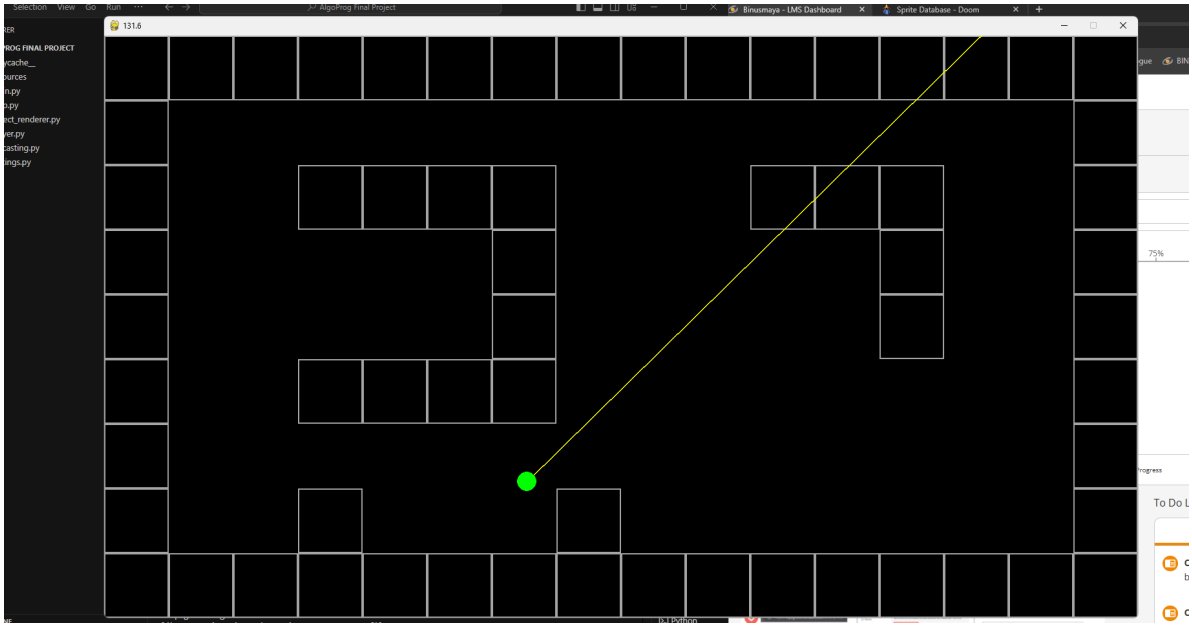


Chapter 3

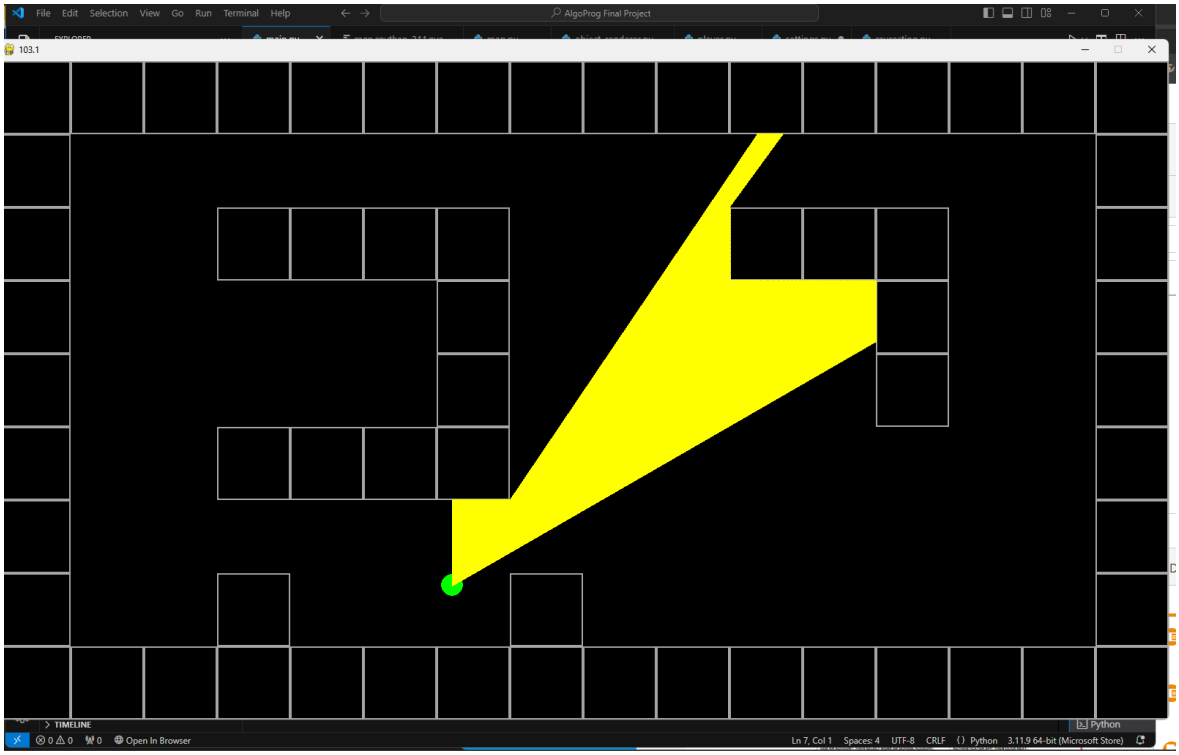
DOCUMENTATION

3.1. Screenshots

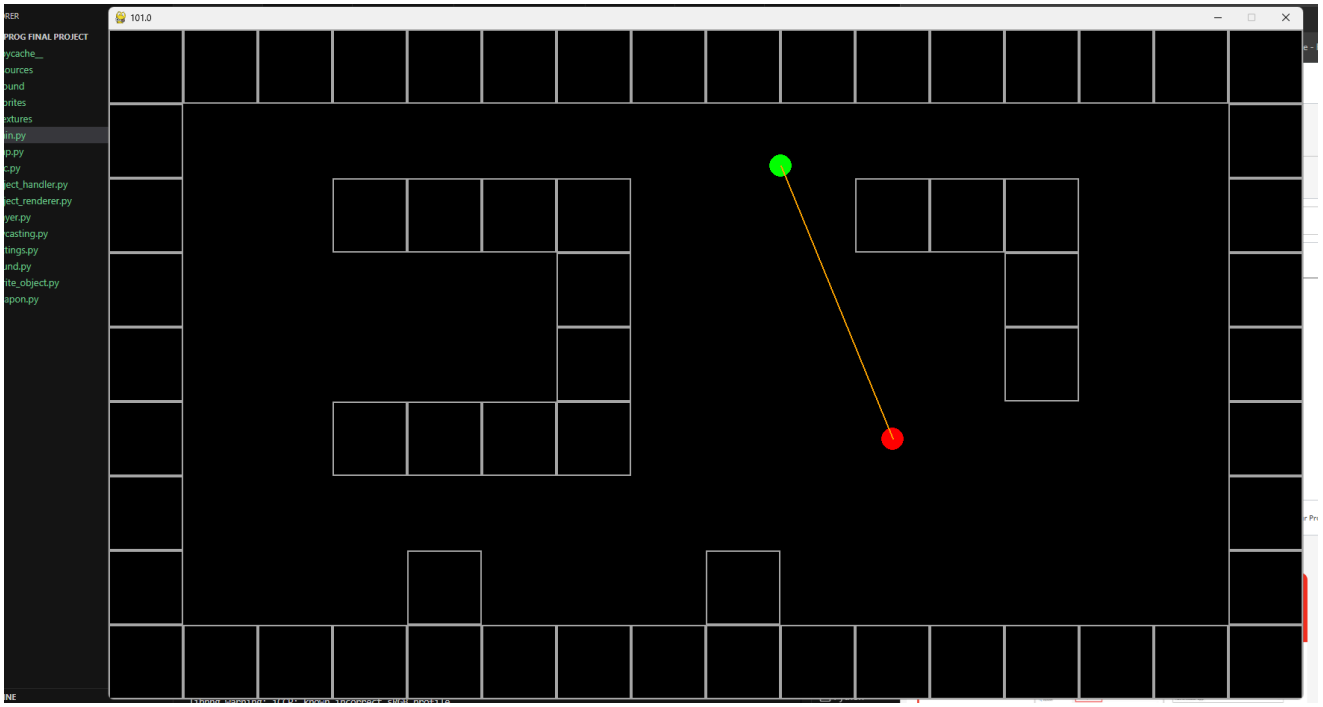
2D Map:



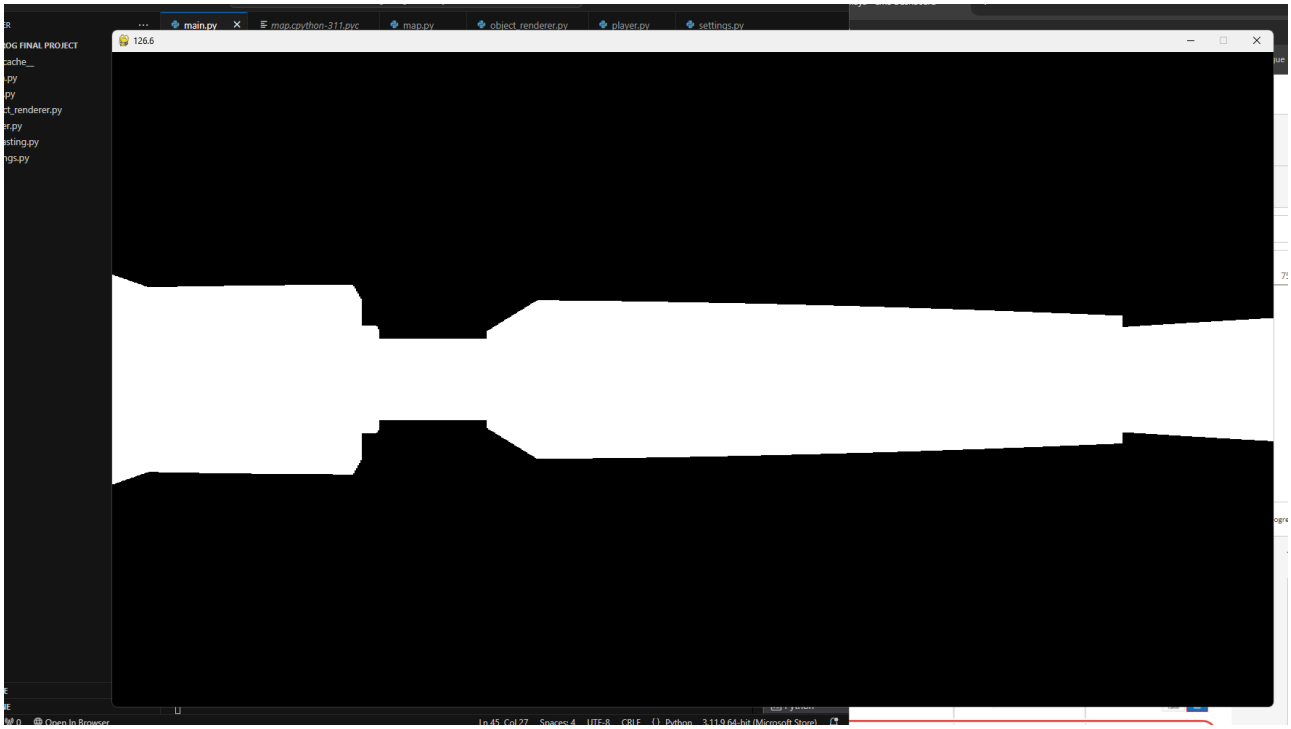
2D Map with ray casting:



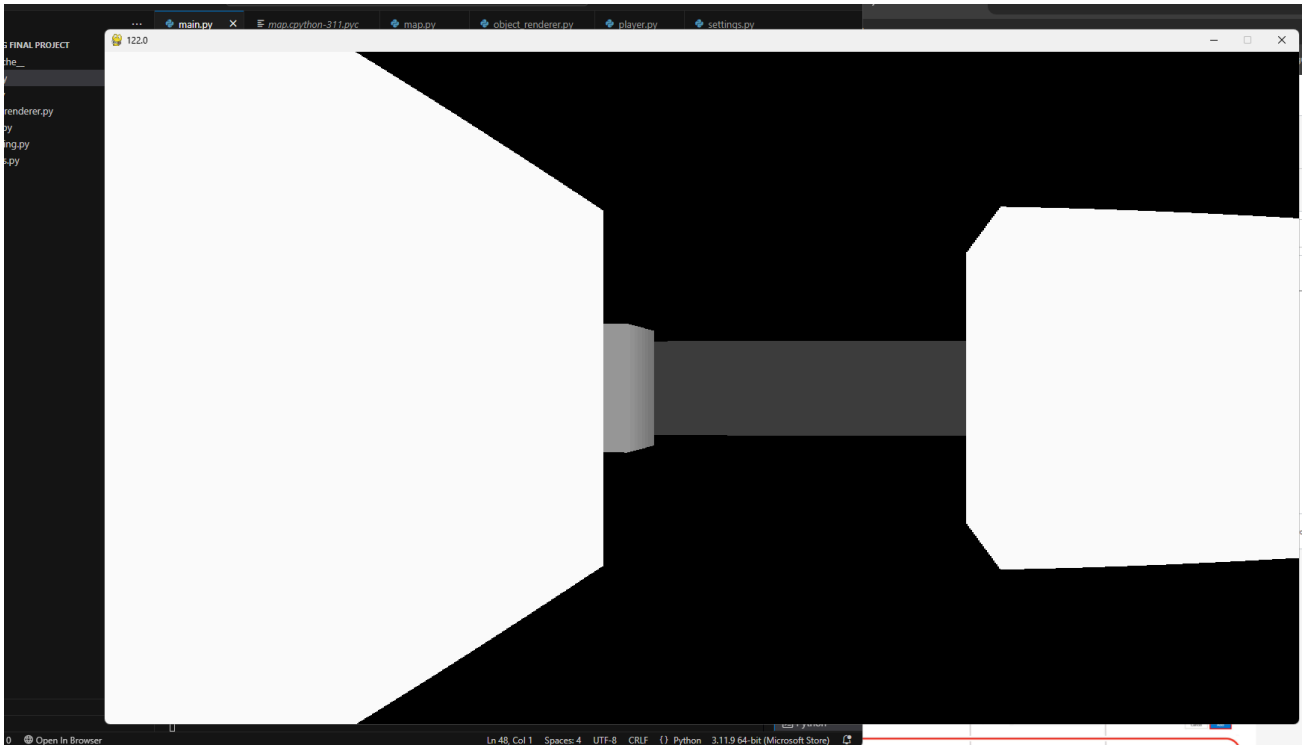
2D Map with enemy tracking player:



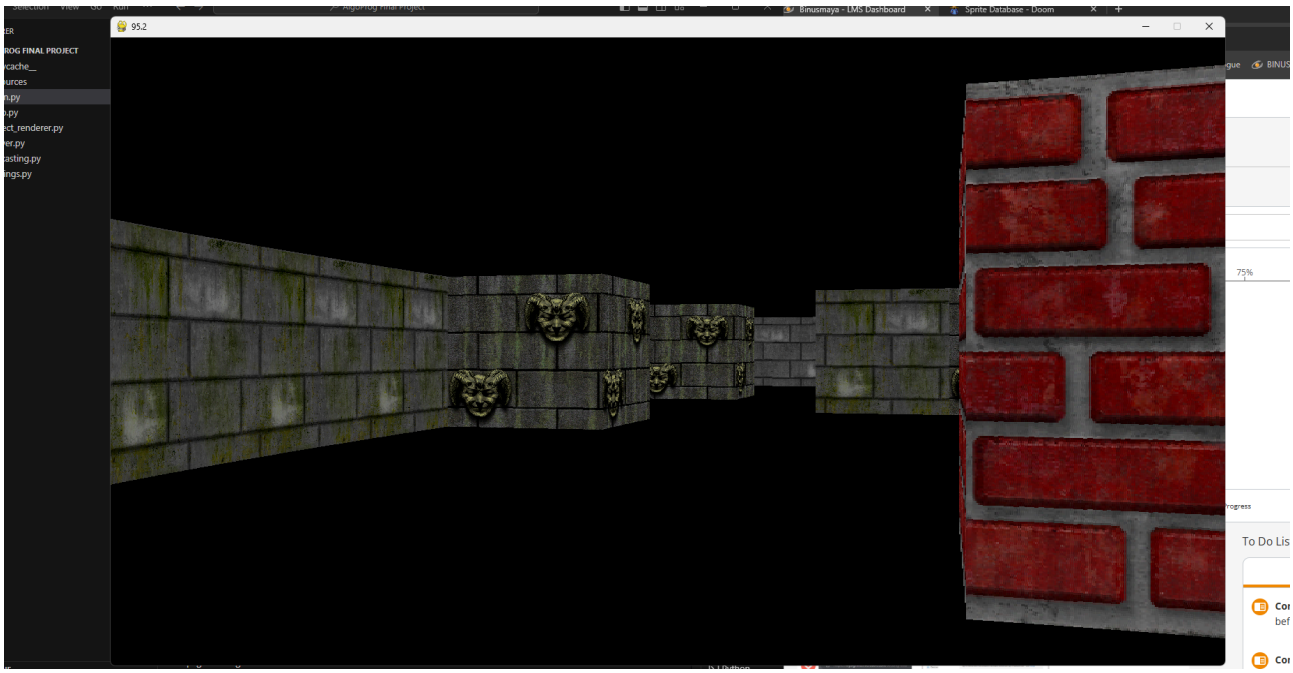
3D Projection:



3D Map with depth:



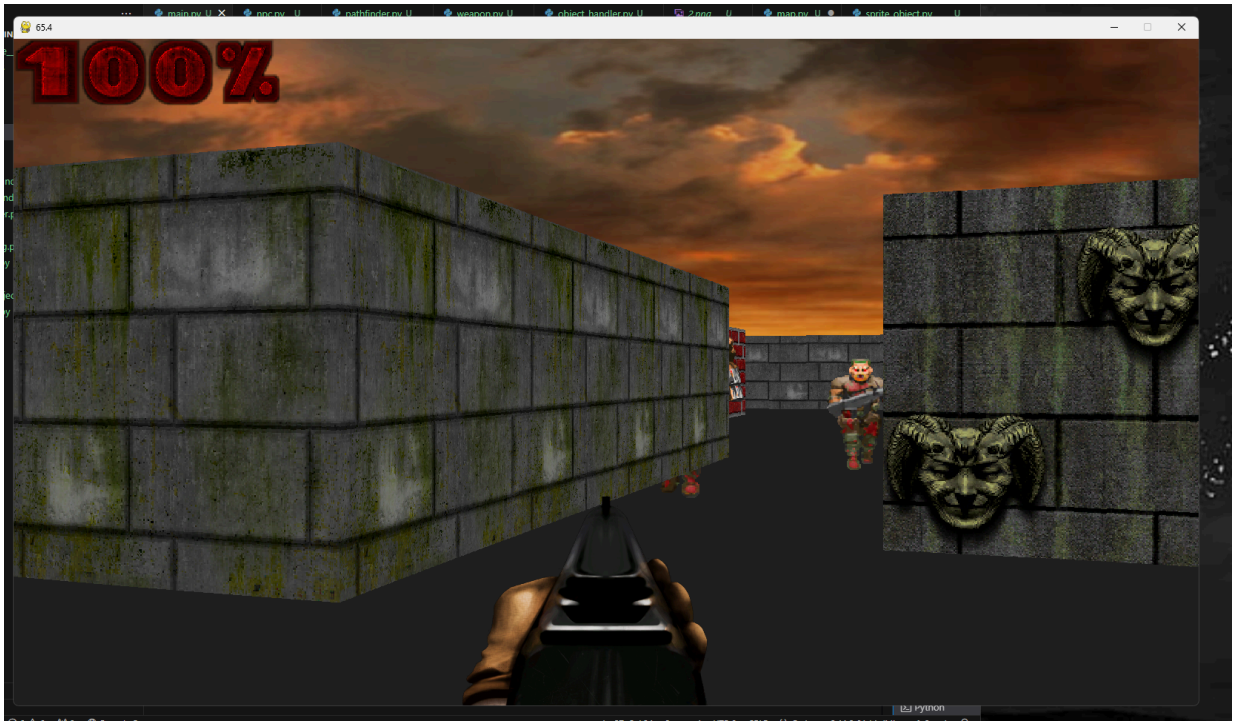
3D Map with wall textures:



Full 3D with 2D props:

Weapon view model & enemy pain animation:

First Game Test:



3.2. Video Demo

I have uploaded a video demo in my Google Drive, which can be viewed through the following link:

Video Demo Link

Chapter 4

EVALUATION AND REFLECTION

4.1. Lessons Learnt

I have learnt a lot of things from this final project. While working on my final project, I learnt a lot of new modules that weren't taught to me in class such as the os and sys modules. I have also found new ways to think of new ideas when implementing certain features into my game.

This final project took around a month more or less to finish with a lot of times of me burning the midnight oil. Through these nights, I have learnt the basic fundamentals of programming games on PyGame such as the game states of the game, the AI level for npcs, making event handlers, using classes to separate different features of the game and others in order to make a completely functional game.

A few important things I have learnt aside from programming my game are time management and never underestimating my task. In the first week, I did not make that much progress because I had another concept for my final project before the current one. It was supposed to be a simple yet fun puzzle game, inspired from a certain popular indie game. Unfortunately as I was working on it, I realised this concept wasn't compatible with what PyGame can provide with its commands. Because of this, I had less than a month to work on my final project. Thankfully, the mid exams help extend the duration of the final project, giving me more time to work on it. In other words, this project has taught me a valuable lesson on not to underestimate my work as well as to manage my time wisely.

4.2. Future Improvements

There are a lot more features in the original Doom 1993 game that aren't available in my game due to the limited duration that was given for the final project. While the game does have its basic elements (enemies, a win/lose ending, player controls), this game doesn't have a main menu and instead, instantly starts the game the moment it's played.

There aren't any ways for the player to modify their settings to fit their playstyle in the game since the game has fixed settings and there is only a single level in this game instead of multiple levels like in the original game.

For the enemies in the game, one of the enemy types was originally supposed to be able to fly but due to the limits of what PyGame can offer, that specific enemy type will only move on

the ground like the others. And because of the limited period given, I didn't have enough time to make projectiles from the enemies' attacks as well as from the player's weapon.

To sum it up, there are a lot of things that can be improved in my game but despite that, I am currently satisfied with what I can do with my best efforts within a tight schedule. Maybe in the future if there is another project that is related to this, I can use this opportunity to improve my game and make it better with adding a menu, options for players to modify their playstyle and more levels to play and keep the players engaged as long as I have the motivation to do so.