ALGORITHM AND PROGRAMMING FINAL PROJECT REPORT

Michael Arianno Chandrarieta

Binus University International Program

Computer Science - 2802499711

JUDE JOSEPH LAMUG MARTINEZ, MCS

26 November 2024

TABLE OF CONTENT

CHAPTER 1

PROJECT SPECIFICATIONS

1.1. Project Description

For my final project, I attempted to recreate a bullet hell game, Project Touhou: Mountain of Faith, one of ZUN's most popular works. Initially, I had planned on making a simple space invaders game, however, due to curiosity and temptation, I decided to recreate Touhou in Pygame, which i then name "A Witch's Hell"

The Witch (player), is a movable entity that is connected to the up, down, left, and right keys. It can shoot and summon bombs to defend itself. The enemies, which I call fairies, are entities that will spawn based on the level difficulty which increase over time. The enemies, similar to the player, are also able to shoot. When spawned, they will shoot bullets with patterns according to the player's location. The goal of the game is to survive and collect points for as long as you can without dying or the game ending. There is no 'winning' situation. Only 'How far can you go?'.

The player is given many ways to survive. First of all, they can shoot back to try and kill the enemies. Second, they are able to summon bombs which clear all projectiles present in the screen, allowing them a 'fresh' screen. Third, they are able to enter 'focus mode' where their shots becomes more precise and deadlier to the enemies. Finally, last of all, they are given 3 lives and 3 bombs, which can also be picked up if the player are running out of it.

Algorithm and Programming Final Project Report

1.2. Project Link

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

https://github.com/MichaelFirstAC/A-WITCH-S-HELL.git

1.3. Essential Algorithm

The essential algorithm of the game "A WITCH'S HELL" can be broken down into several key components:

1. Initialization

- Import and Initialize: The game imports necessary modules (pygame, game_sprites, random) and initializes Pygame. This sets up the environment for the game to run.
- Pre-initialize Sound: The pygame.mixer.pre_init function is used to reduce sound delay, ensuring that sound effects play more responsively.
- Initialize Display: The game window is set up with a specific resolution, and the game icon and caption are set.

2. Main Function

- Main Loop: The main function sets up the main menu loop and handles transitions between different game states (main menu, game loop, pause, game over).

3. Game Intro (Main Menu)

- Display Menu: The game_intro function loads and displays the main menu background and buttons. It allows the player to navigate through menu options using keyboard inputs and select options to start the game, view the leaderboard, erase data, or quit.
- Play Background Music: Background music is played in a loop, and a random track is selected each time the game starts or when a track ends.
-

4. Game Loop

- Entities Setup: The game_loop function creates and initializes game entities such as the player, enemies, bullets, score tab, and background.
- Event Handling: The game handles player inputs for movement, shooting, bombing, and pausing the game.
- Game Logic:
    - Player Actions: The player can move, shoot, and use bombs. The player's position and actions are updated based on input.
    - Enemy Actions: Enemies are spawned, move, and shoot at the player. The enemy behavior varies based on their type.
    - Collision Detection: Collisions between the player, enemies, bullets, and pickups are detected and handled.
    - Score and Lives: The player's score and lives are updated based on game events.
    - Difficulty Adjustment: The game difficulty increases over time by adjusting enemy spawn rates and limits.
- Update and Render: All game entities are updated, and the screen is redrawn.

5. Pause Function

- Display Pause Screen: The pause function displays a darker paused frame with options to resume or return to the main menu.
- Handle Input: The player can navigate through pause menu options using keyboard inputs.

6. Game Over Function

- Display Game Over Screen: The game_over function displays a darker game over frame with options to restart or return to the main menu.
- Handle Input: The player can navigate through game over menu options using keyboard inputs.

7. Music Handling

- Random Music Selection: The game randomly selects and plays a background music track from a list each time the game starts or when a track ends.
- Music End Event: The game detects when a music track ends and plays a new random track.

8. Save and Load Data

- High Score: The game saves the high score to a file when the game ends and loads it when the game starts.
-

Algorithm and Programming Final Project Report

Button Class

- Initialization: The Button class initializes a button sprite with a position, message, and color. It sets up the font and renders the initial button image.
- Selection Handling: The set_select method sets the button as selected, and the update method changes the button color if it is selected.

Player Class

- Initialization: The Player class initializes the player sprite with various animation frames, position, and movement properties.
- Movement and Shooting: The change_direction, diagonal_mode, focus_mode, and shoot_mode methods handle player movement and shooting modes. The spawn_bullet method spawns bullets based on the player's shooting mode.
- Reset and Invincibility: The reset method resets the player's position and state, and the set_invincible method sets the player to be invincible for a certain number of frames.
- Update: The update method updates the player's animation, movement, and invincibility state.

Enemy Class

- Initialization: The Enemy class initializes enemy sprites with different properties based on their type. It sets up animation frames, movement vectors, and health.
- Spawning and Shooting: The setup method sets the enemy's spawn position, and the spawn_bullet method spawns bullets aimed at the player.

- Damage and Death: The damaged method decreases the enemy's health, and the set_killed method marks the enemy as killed.
- Update: The update method updates the enemy's animation, movement, and shooting cooldowns.

Bullet Class

- Initialization: The Bullet class initializes bullet sprites with different properties based on their type and direction.
- Update: The update method updates the bullet's position and checks for collisions or if the bullet is out of bounds.

Score_tab Class

- Initialization: The Score_tab class initializes the score tab with the player's score, high score, lives, and bombs.
- Update: The update method updates the score tab's display based on the player's current score, lives, and bombs.

Summary

The game algorithm involves initializing the game, handling the main menu, running the game loop with player and enemy interactions, handling pause and game over states, and managing background music and high scores. The game loop is the core of the game, where most of the gameplay logic, including player actions, enemy actions, collision detection, and rendering, takes place. The game_sprites.py file defines various sprite classes that represent different game entities and their behaviors.

1.4. Modules

1. PyGame:

The foundation of the entire game. Without PyGame, the game would only be a terminal/text-based game. PyGame allowed me to make the GUI for the game by blitting images onto the screen and using the mixer to play audio in order to make the game more lively. Without PyGame, most of the time-based mechanics would not work either.

2. Math Module:

The Math Module plays an important role in this game. It handles angle calculations for bullet directions and ensures consistent movement speeds, which are essential for the gameplay mechanics of a bullet hell game.

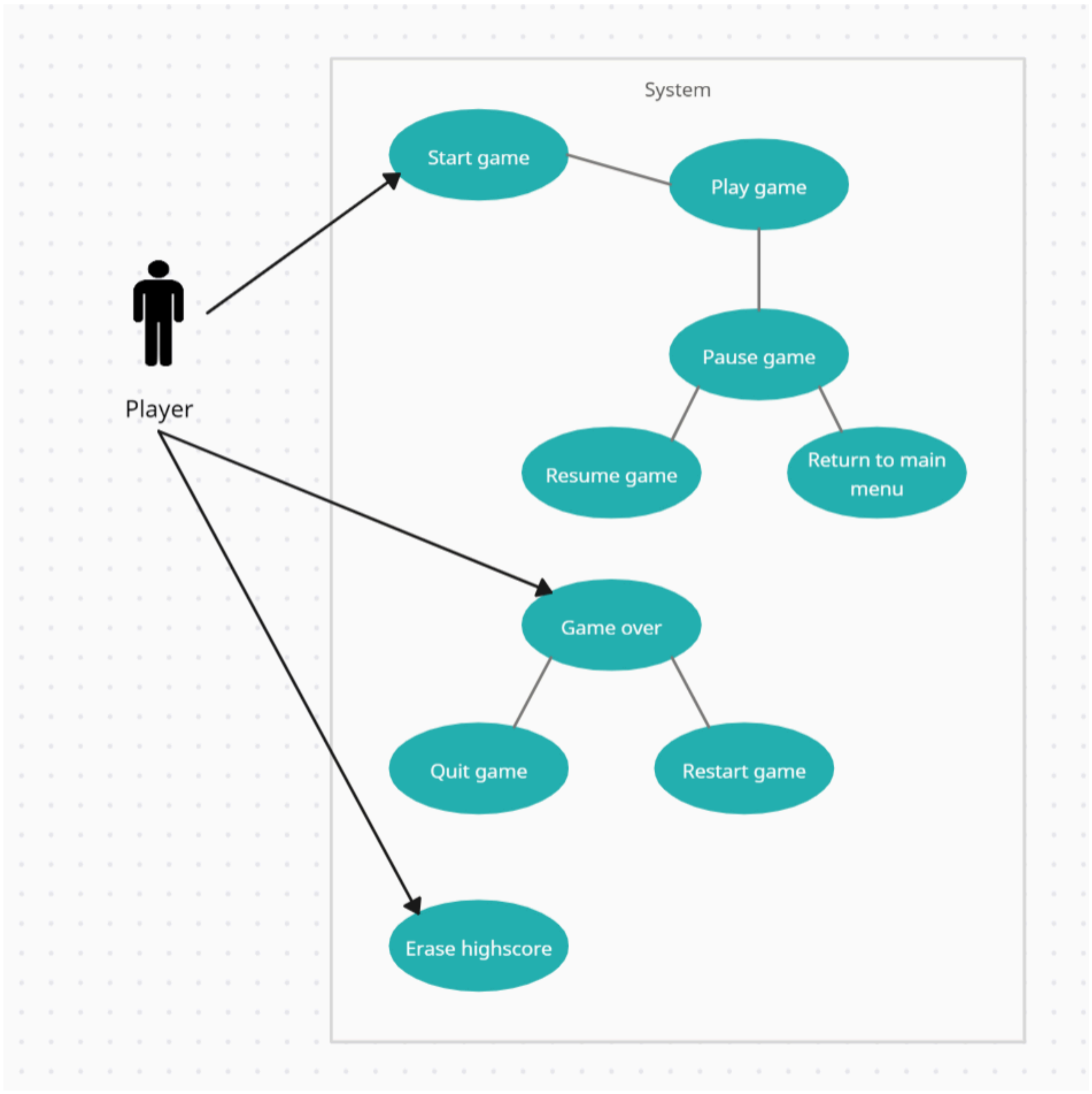
3. Random Module:

The Random Module also plays an important role as it is involved in the randomness aspect of this game. It adds randomness to enemy spawn locations, item drops, and cloud movements, making the game more dynamic and unpredictable.

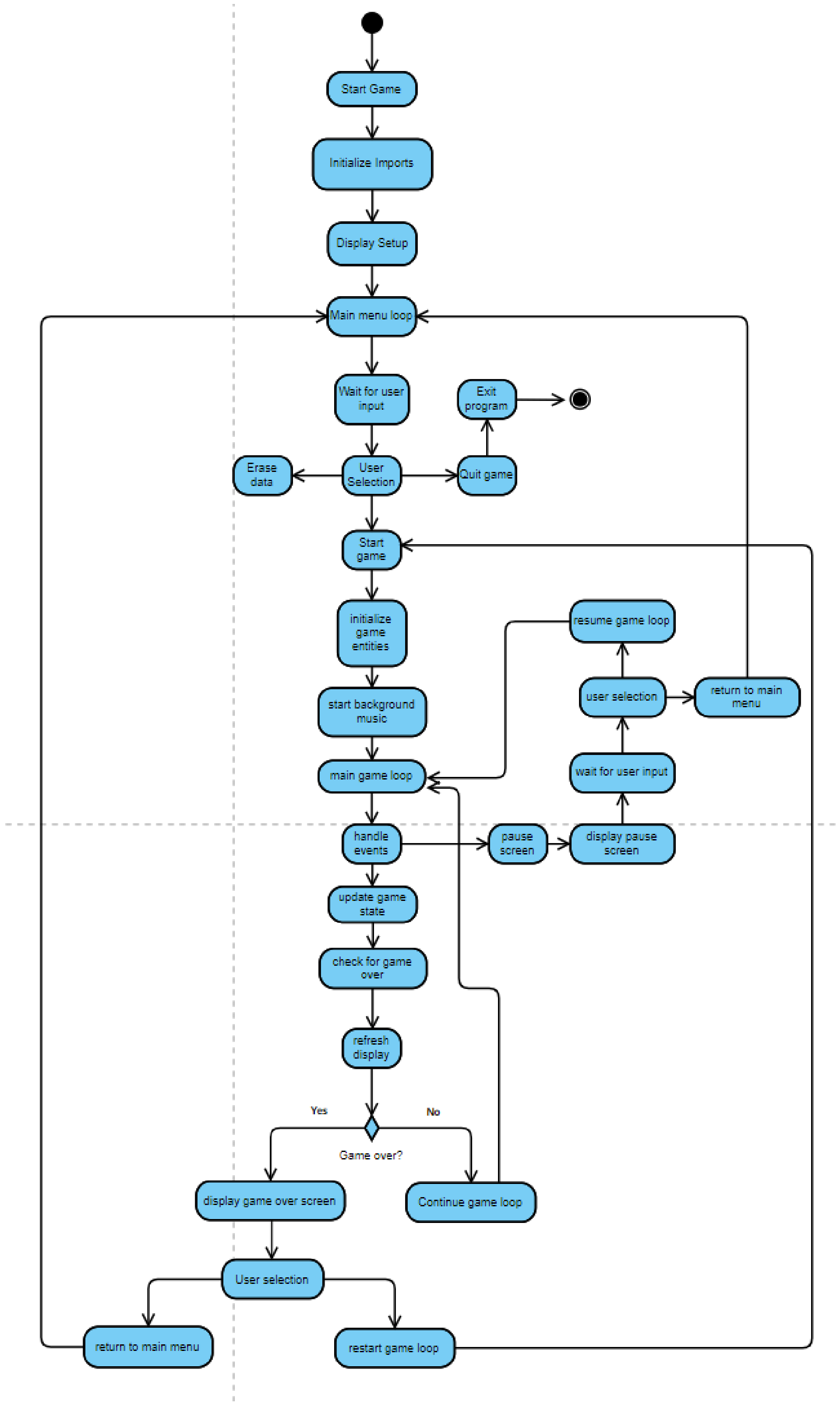
CHAPTER 2

SOLUTION DESIGN

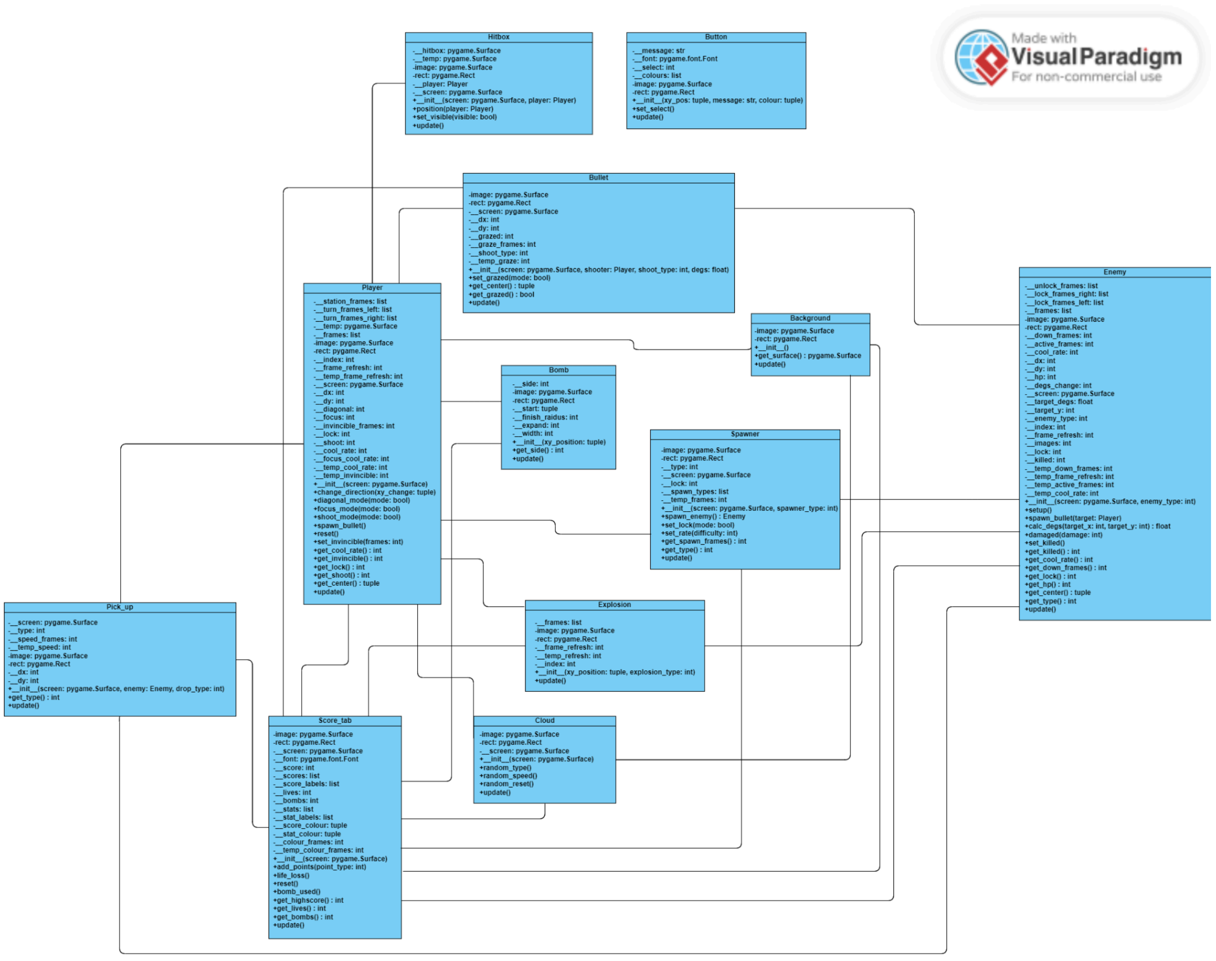
2.1. Use Case Diagram



2.2. Activity Diagram



2.3. Class Diagram



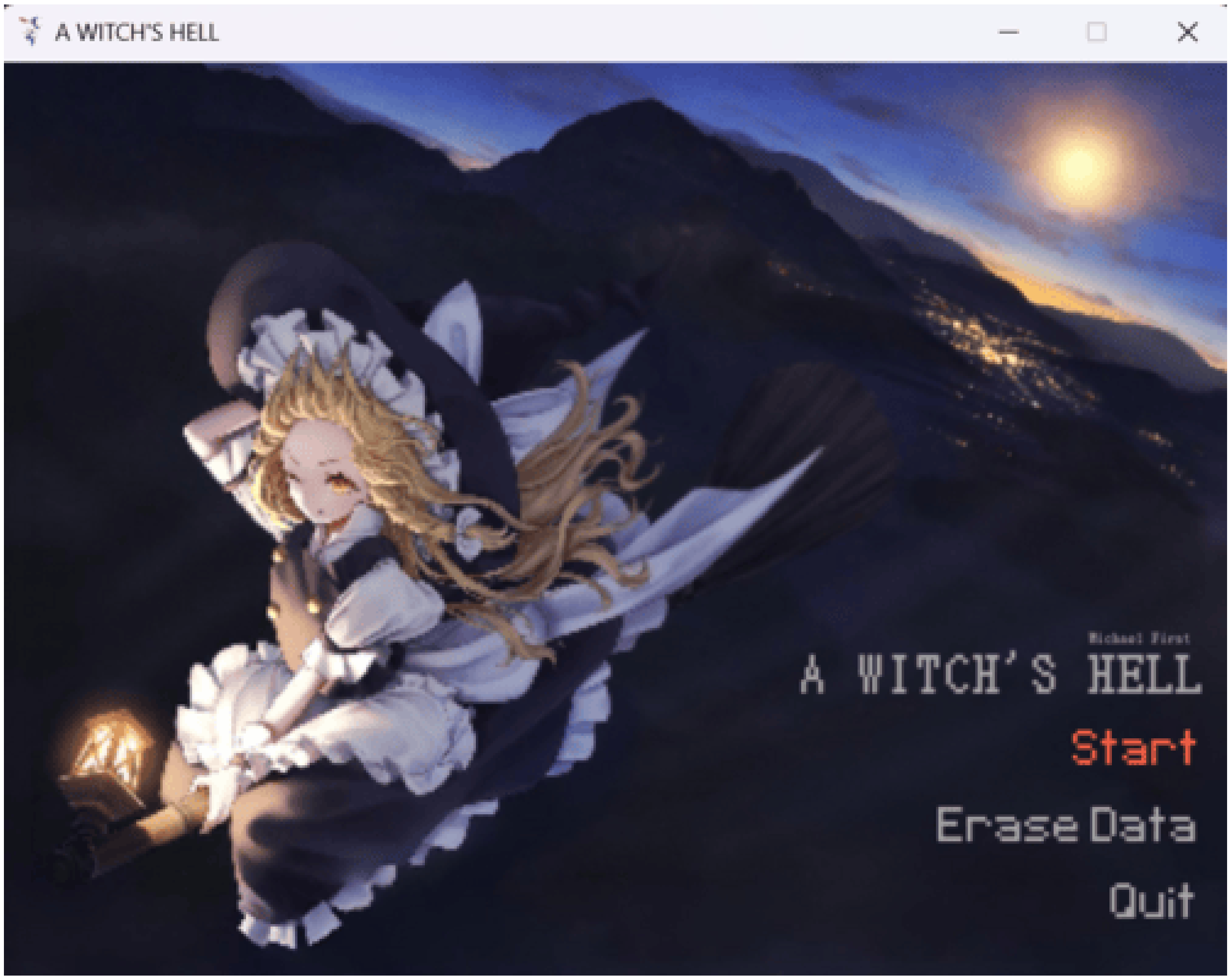
Clearer pictures could be accessed in the Github!

CHAPTER 3

DOCUMENTATION

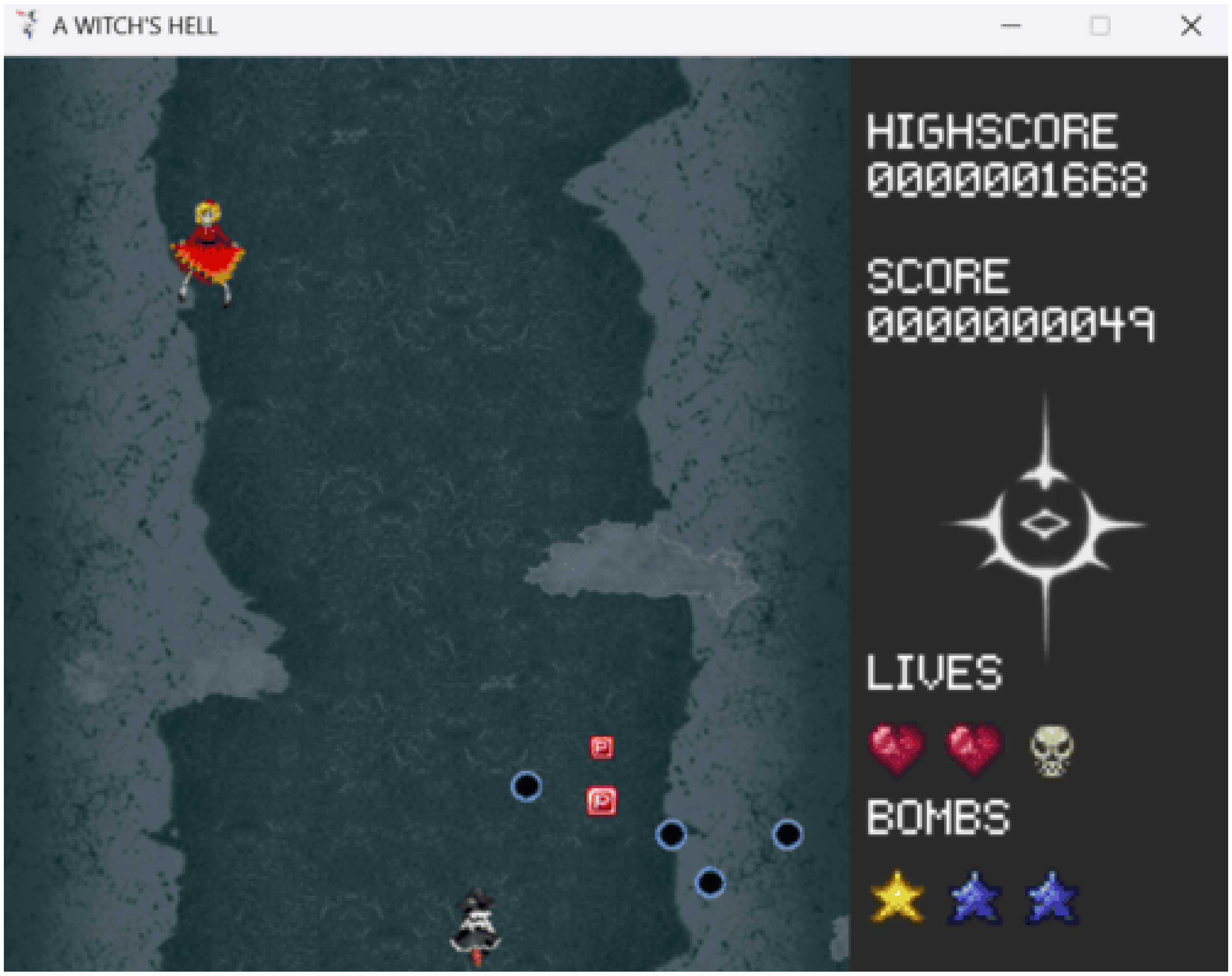
3.1. Screenshots

Main Menu

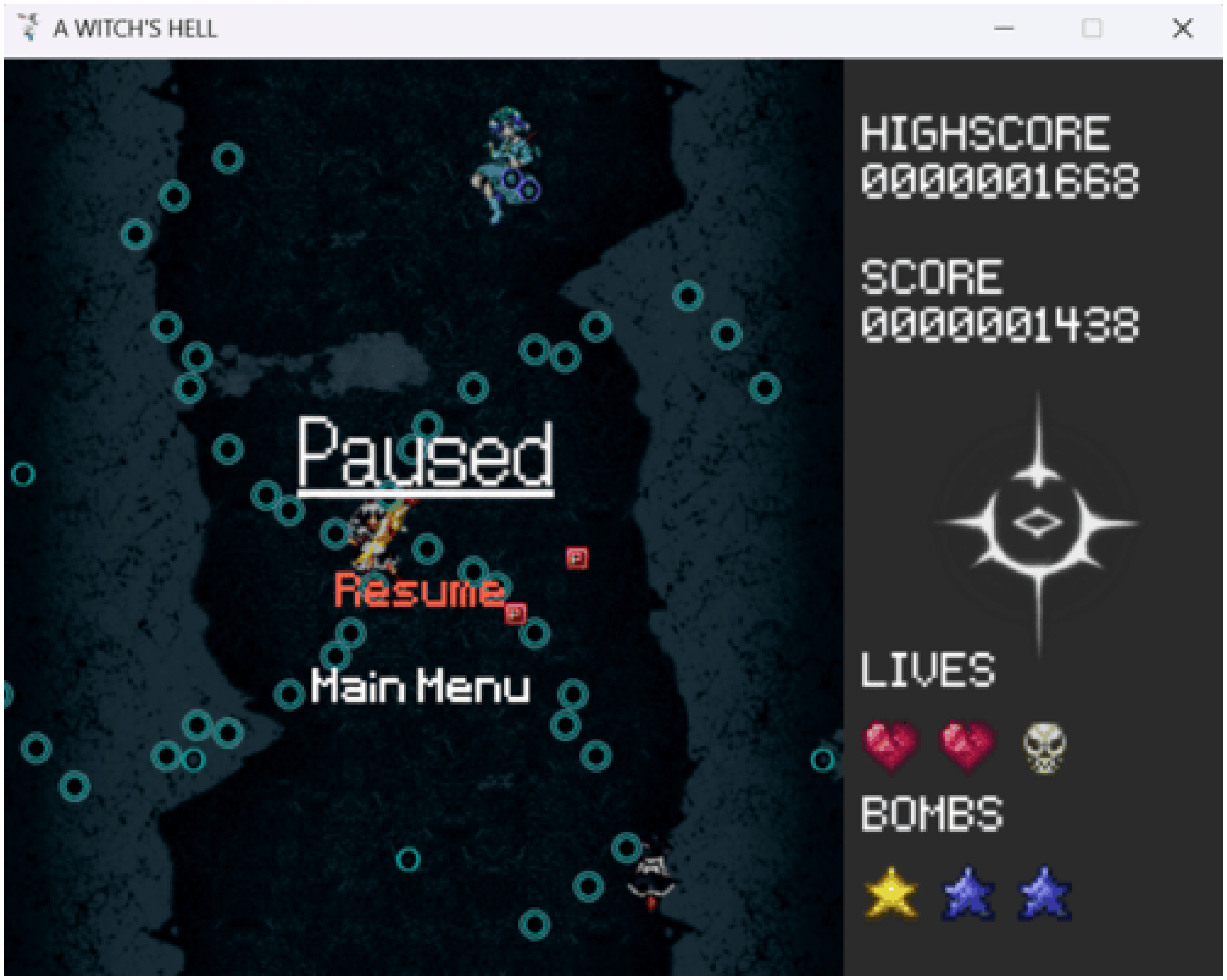


Algorithm and Programming Final Project Report

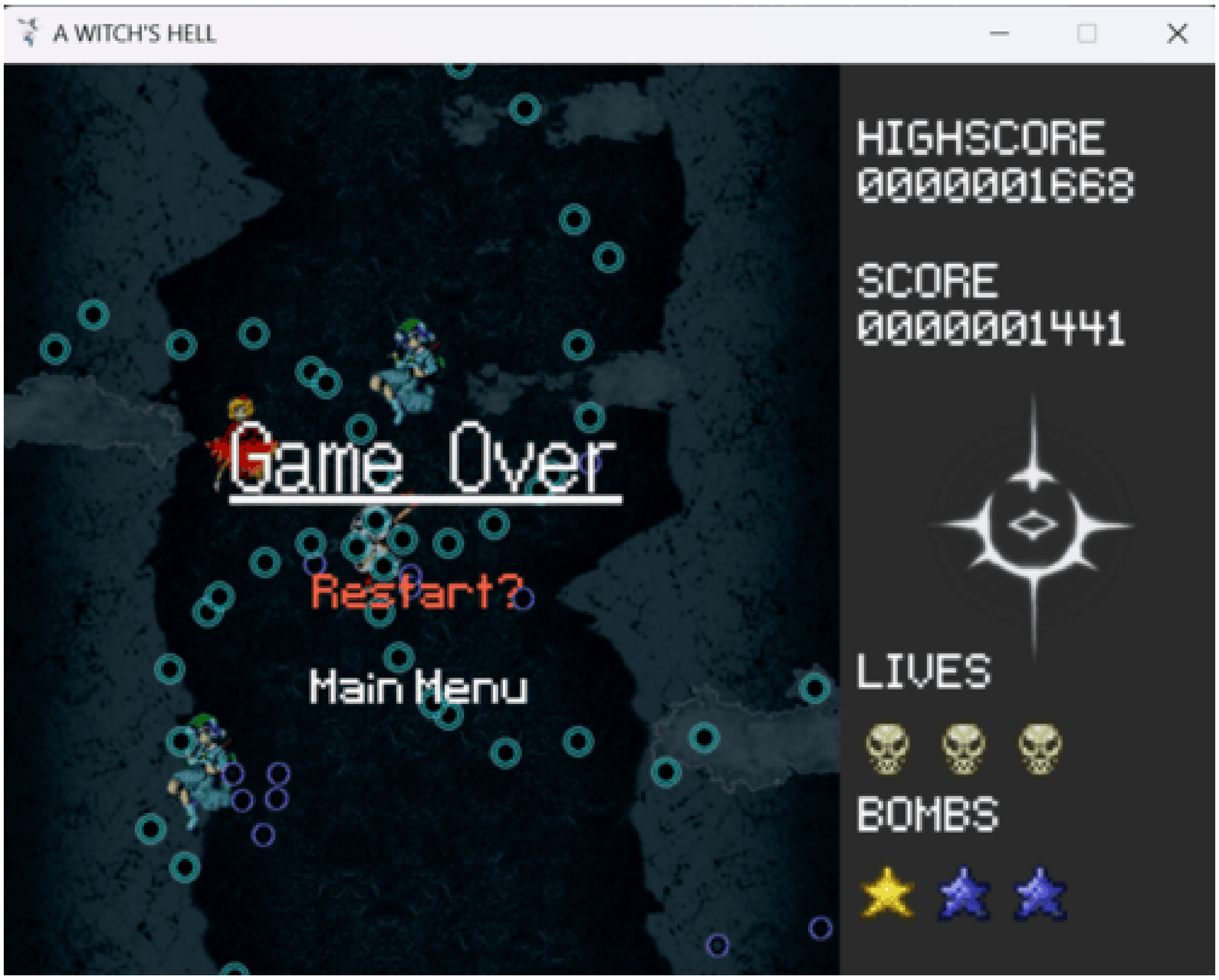
Game Screen



Pause Screen



Game Over Screen



3.2. Video Demo

I have uploaded the video demo to my Google Drive. The video can be accessed through the link is provided below:

https://drive.google.com/file/d/1R480CEgUhq2UXVdb8umG6TU9YWjqFYaW/view?usp=sharing

3.3. References

I would like to give my deepest thanks to this game which i took as a reference for the skeletal structure of my game.

Spess Shooter By Anon1023

https://www.pygame.org/project-Spess+shooter-2405-.html

https://sourceforge.net/projects/spessshooter/

Youtube tutorials:

https://youtu.be/AY9MnQ4x3zk?si=RYILManV8zjVIOcB

https://youtu.be/FfWpgLFMI7w?si=UaEBuj_ipR6zbPsS

And of course, I would like to thank ZUN for the assets I had used in my game, and for the idea of my game. Project Touhou: Mountain of faith is a favorite game of mine, and being able to remake somewhat of it, i am quite happy of.

CHAPTER 4

EVALUATION AND REFLECTION

4.1. Lessons Learnt

I learned a lot of new things when I was working on my final project. Not only did I realize that Python could be used as a game-development language, but I also how much I despise coding AI elements of the game.

I have stayed up until the sun rises to get this project done before the day of the submission and picked up some nice valuable insights/lessons along the way. I have learned that being curious is both a good and bad trait to have. The amount of time I was in for being stuck in a single section of my code, specifically the enemy bullets, entities, and spawning algorithm. I had to watch a lot of YouTube tutorials just to get my things together.

Aside from this, I have also come to be reminded of the importance of ‘passion’. Without passion, I would dare say that my work will either not be done, or is something completely boring in my eyes. If my passion for games and coding in general wasn't apparent, this project would be something I am not quite proud of. I was already expecting that coding a bullet hell game would be hard, but I severely underestimated how ‘hard’ it is. It was at least 2x as hard as I originally thought, so I'm thankful I decided to start on this project in October. This project serves as a lesson and reminder for me, that as long as there is a drive, a passion, and a will for me to actually enjoy or be interested in coding, I will do just well.

4.2. Future Improvements

There are a lot of features in the original Touhou game that I am not able to implement in this game. For example, levels and a super boss-type enemy. I was planning to implement a super boss enemy in the game each time the game reaches a specific amount of points, but, sadly, i am not able to do so without crashing the game altogether.

Levels were not implemented purposefully. Considering that if I were to use a level system in the game, it means I have to code MORE enemies and make a ‘super boss’ on each level to reach a climax. So, to get a genuine ‘hell’ like experience in this bullet hell game, I decided to make it endless with a difficulty spike. The longer the game runs, the harder it gets. At some point, you will experience genuine hell when 2 boss-type 3 enemies spawn at the same time.

Some other things that I could’ve implemented but felt unmotivated to do so are making more and more enemies. However, doing so would mean I had to change some aspects of my original code to fit it, and by the time I had that idea, I was too busy with other projects, so I decided otherwise and stuck to the original 5 enemies.

Overall, I believe there could be so much more room for improvement and things to be implemented in my project. But even knowing fully well about this, I am very satisfied with what I was able to make within such a time frame, even if it is only a remake of an already existing game and not at all an original idea with original assets.