ALGORITHM AND PROGRAMMING FINAL PROJECT REPORT

Jason Franto Fong

Binus University International Program

Computer Science - 2802557781

JUDE JOSEPH LAMUG MARTINEZ, MCS

8 January 2025

TABLE OF CONTENT

CHAPTER 1

PROJECT SPECIFICATIONS

1.1. Project Description

My final project is called 'Survivor' pretty simple I know, but the aim of the game is just like its title, it is to survive. Initially, I had 3 ideas or concept that I wanted to use for my final project. The first one was making a simple rogue like game with POKEMON aspects to it but as I kept on working on it, I scrapped it due to the lack in confidence of not being able to finish it. The second idea was in making a platformer game with multiple inputs such as dashes and etc., I had made the game halfway but decided that it was not satisfactory as there were to much bugs. The final idea was the concept I currently have now and have presented.

The game is about a singular survivor where the survivor has to survive being attacked by hordes of enemies with just a singular pistol to keep them at bay. The survivor has usual movement inputs such as up down left right, and are able to shoot bullets with the gun using input from the mouse cursor. The enemies in this case, they have no ranged weapons and only deal damage by colliding with the survivor, this is the only way they can deal damage. But not to worry, in any spot the survivor is located, every enemy or entity will be attracted to it making the game harder.

The goal of the game is for the survivor to kill 50 enemies so the survivor can escape safely. This task is pretty simple as the survivor would just have to dodge these enemies or entities and kill enough to finish the game.

Algorithm and Programming Final Project Report

1.2. Project Link

The GitHub Repository of the project: https://github.com/Jasonnnnnnn1/AlgoProg-Final-Project

1.3. Essential Algorithm

The essential algorithm of the game "Survivor" can be broken down into several key components:

1. Game Initialization and Setup

- Initializes the game environment, sets up the display, and loads assets.
- __init__ Method:
  - Initializes Pygame, sets up the display window, and configures the game clock.
  - Creates sprite groups (all_sprites, collision_sprites, bullet_sprites, enemy_sprites) for managing game objects.
  - Loads audio files and sets up timers for enemy spawning and shooting cooldowns.
- load_images Method:
  - Loads bullet and enemy animations from specific directories.
  - Organizes enemy frames into a dictionary for easy access during gameplay.
- setup Method:
  - Resets the game state by clearing sprite groups.
  - Loads the game map from a .tmx file and populates the game world with ground tiles, objects, collisions, and entities (player and spawn positions).

2. Player Input Handling

- Processes player input for shooting bullets.
- input Method:

  - Checks if the left mouse button is pressed and if the player can shoot (based on the cooldown timer).
  - Plays a shooting sound and spawns a bullet at the player's gun position, moving in the direction the player is facing.
  - Resets the shooting cooldown timer.

3. Shooting Cooldown Management

- Ensures the player cannot shoot continuously by enforcing a cooldown period.
- gun_timer Method:
  - Tracks the time elapsed since the last shot.
  - Enables shooting again once the cooldown period (gun_cooldown) has passed.

4. Enemy Spawning

- Spawns enemies at random intervals and positions.
- enemy_event Timer:
  - Triggers every 2 seconds to spawn a new enemy.
  - Chooses a random spawn position from the predefined list and a random enemy animation from the loaded frames. (predefined list can be accessed through the .tmx file)
  - Adds the enemy to the all_sprites and enemy_sprites groups.

5. Collision Detection

- Detects collisions between bullets, enemies, and the player.
- bullet_collision Method:
  - Checks for collisions between bullets and enemies using pygame.sprite.spritecollide.
  - Plays an impact sound, destroys the enemy, and increments the killed_enemies counter.
  - If the player kills 50 enemies, the game transitions to the "game finished" state.

- player_collision Method:
    - Checks for collisions between the player and enemies.
    - Reduces the player's health if a collision occurs.
    - If the player's health drops to 0, the game transitions to the "game over" state.

6. Game State Management

- Manages transitions between different game states (title screen, gameplay, game over, game finished).
- title_screen Method:
    - Displays the title screen with a background image, title text, and instructions.
    - Waits for player input (key press or mouse click) to start the game.
- game_over Method:
    - Displays a "Game Over" screen with options to restart or quit the game.
    - Resets the game state if the player chooses to restart.
- game_finished Method:
    - Displays a "Game Finished" screen when the player kills 50 enemies.
    - Provides options to restart or quit the game.

7. Rendering and UI

- Draws the game world, health bar, and kill counter on the screen.
- draw_health_bar Method:
    - Draws a health bar at the top-left corner of the screen.
    - The bar's width is proportional to the player's current health.
- draw_killed_enemies_counter Method:
    - Displays the number of enemies killed out of 50 at the top-left corner of the screen.
- all_sprites.draw Method:

  - Draws all sprites on the screen, centered around the player's position.

8. Main Game Loop

- Continuously updates and renders the game.
- run Method:
  - Starts with the title screen and enters the main game loop.
  - Handles events (like quitting the game and spawning enemies).
  - Updates game state by calling methods for input handling, collision detection, and sprite updates.
  - Renders the game world, health bar, and kill counter on each frame.

This structured approach ensures that each aspect of the game is managed efficiently, providing a smooth gameplay experience.

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 unique and interesting. Without PyGame, most of the time-based mechanics would not work either.

2. Sys Module:

Although not used as much as PyGame, the sys module was only used to close the game with the sys.exit) method. Aside from that, nothing else from the Sys module was really utilized.

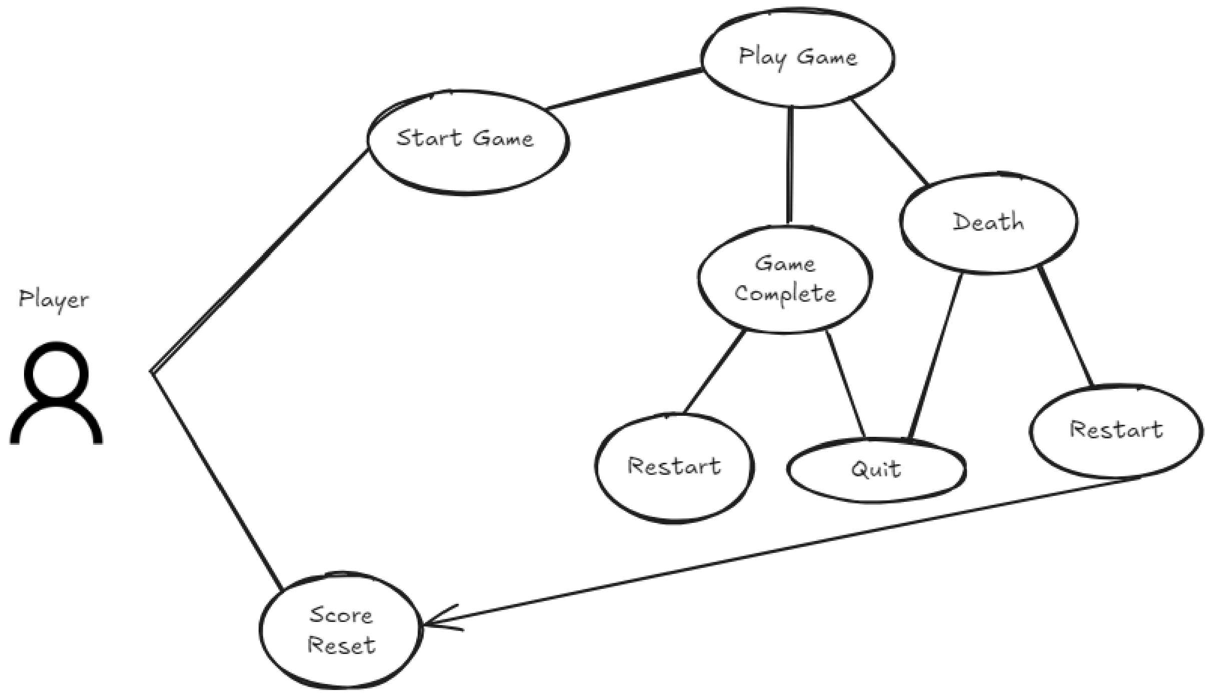
3. Math Module:

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.

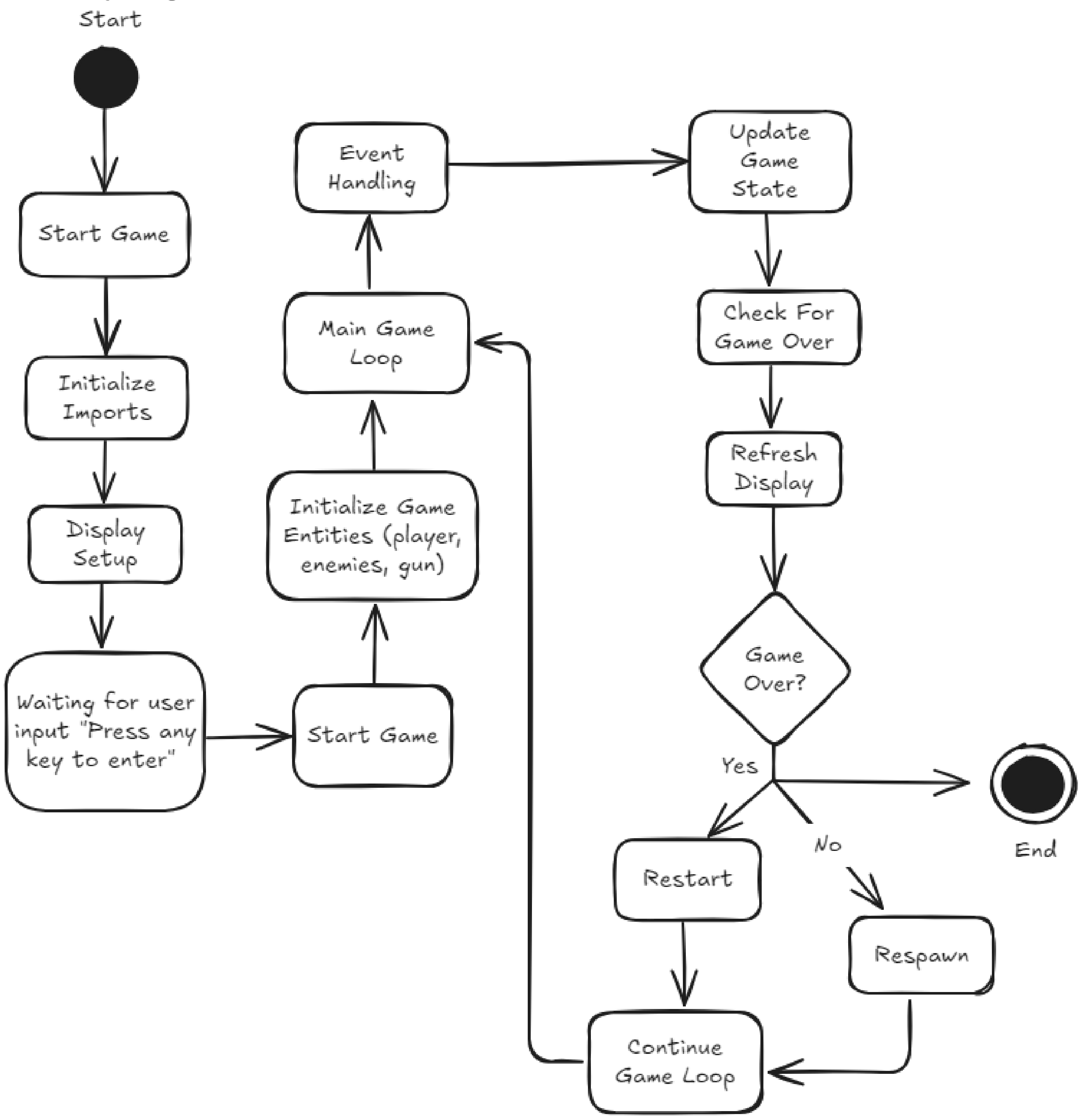
CHAPTER 2

SOLUTION DESIGN

2.1. Use Case Diagram

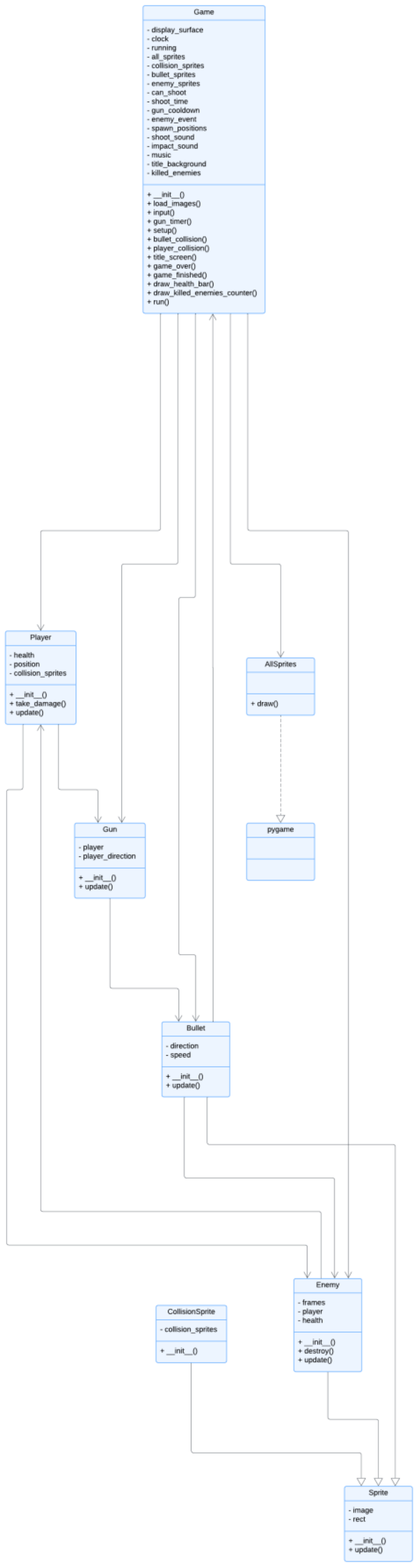


2.2. Activity Diagram



2.3. Class Diagram

Algorithm and Programming Final Project Report

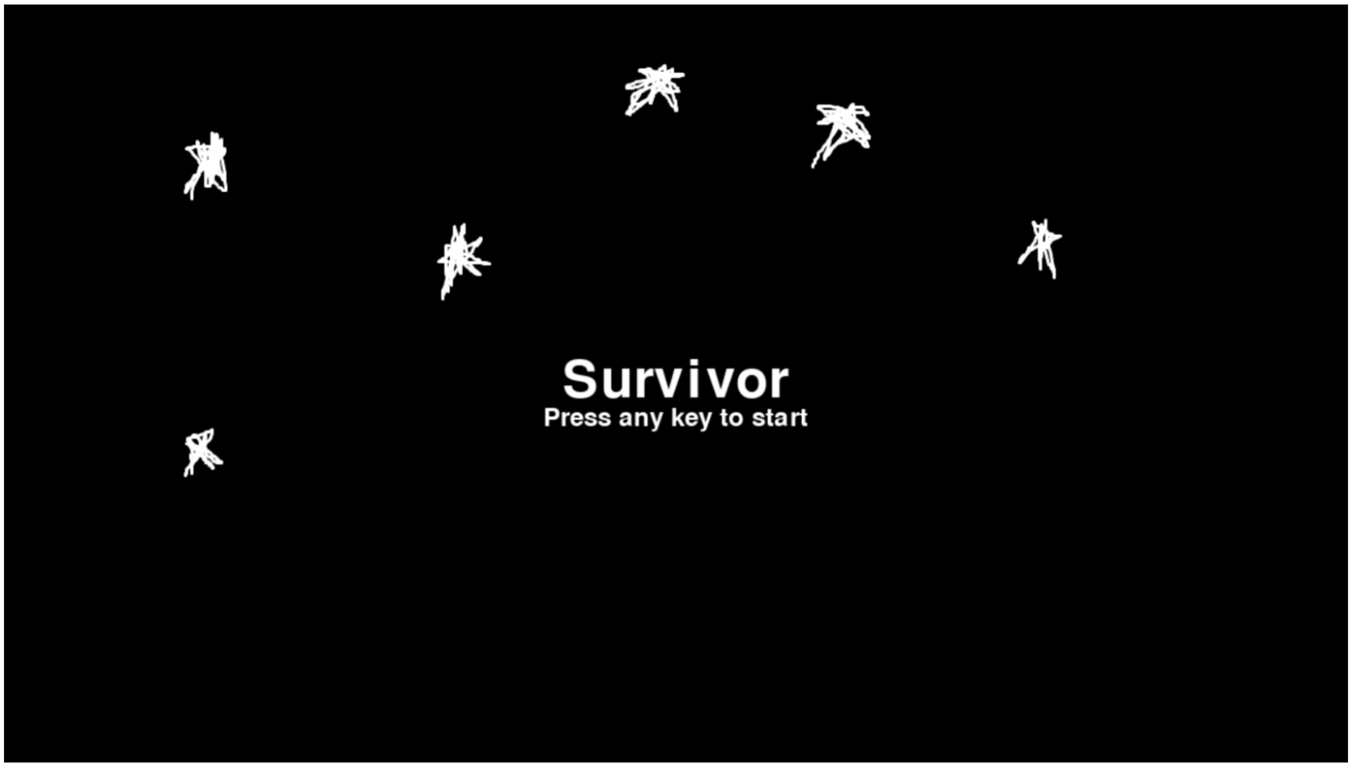


CHAPTER 3

DOCUMENTATION

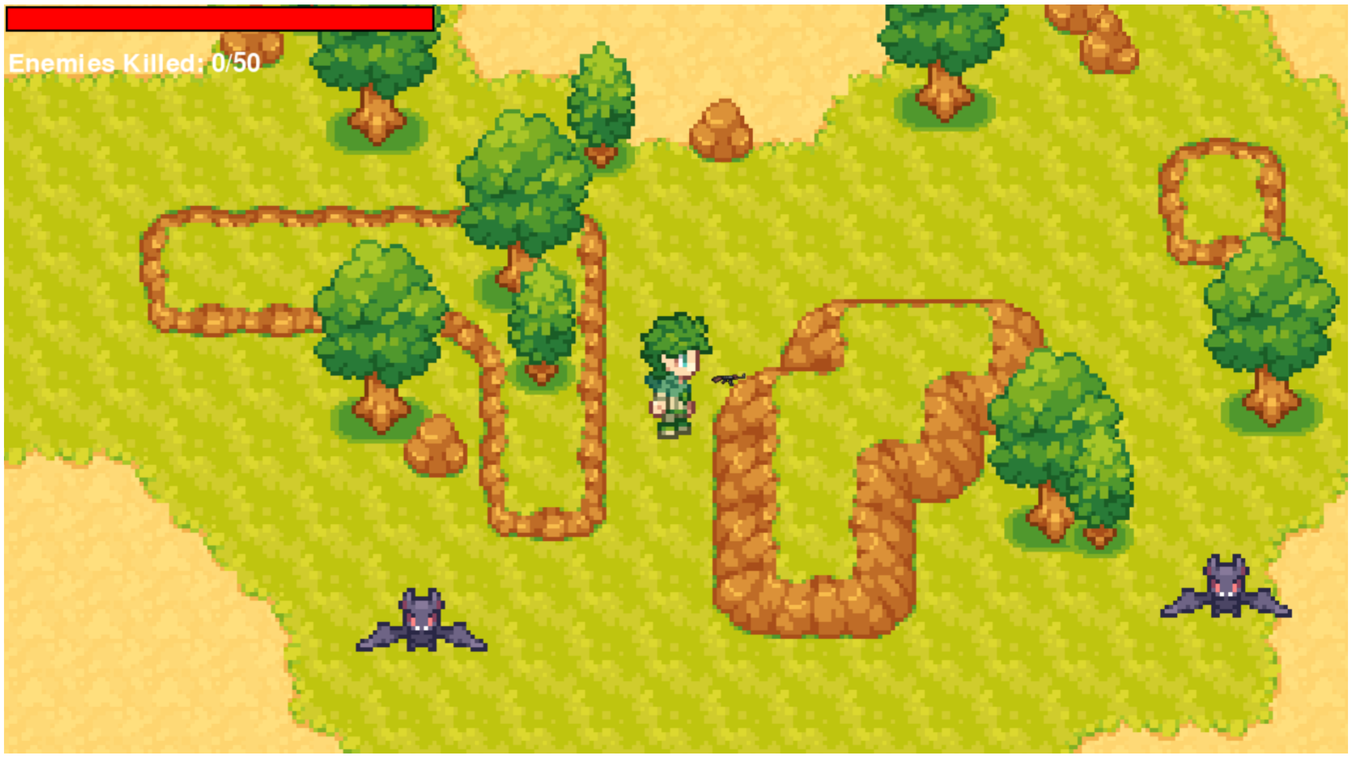
1.1. Screenshots

Main Menu

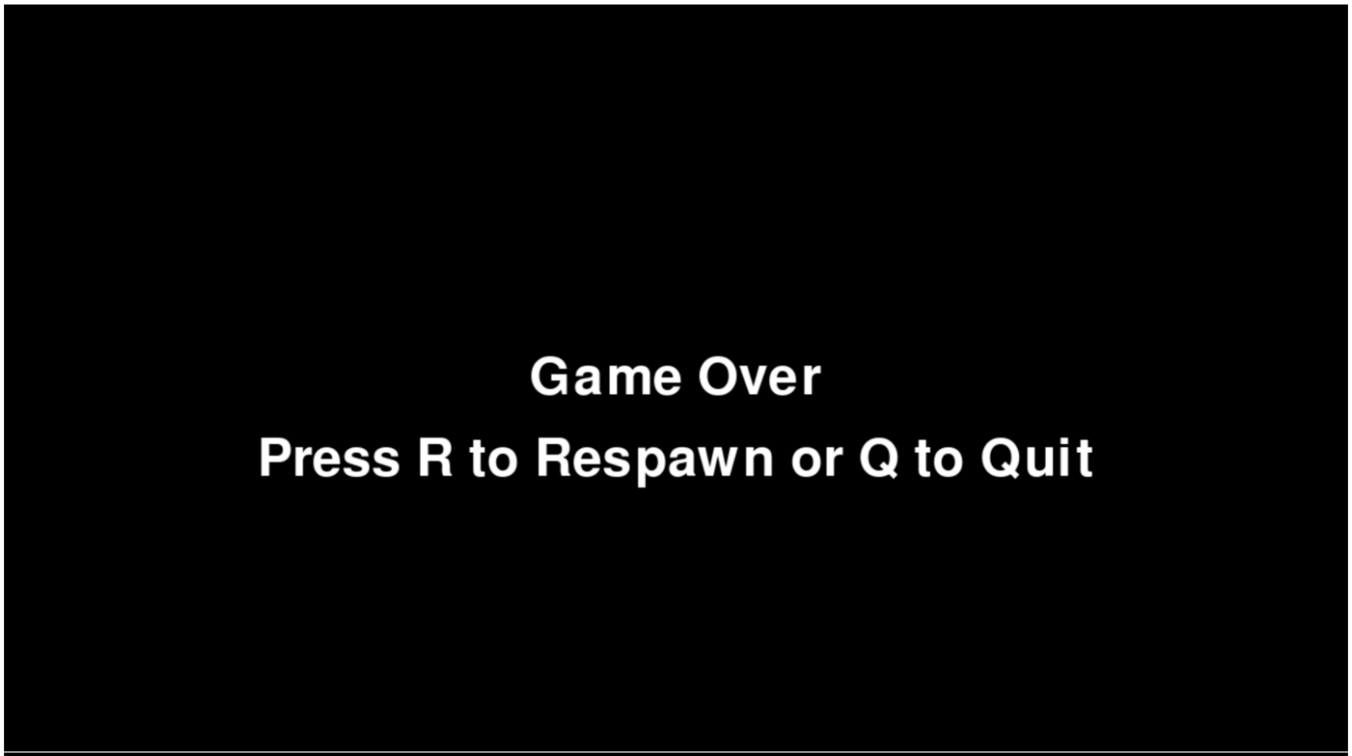


Algorithm and Programming Final Project Report

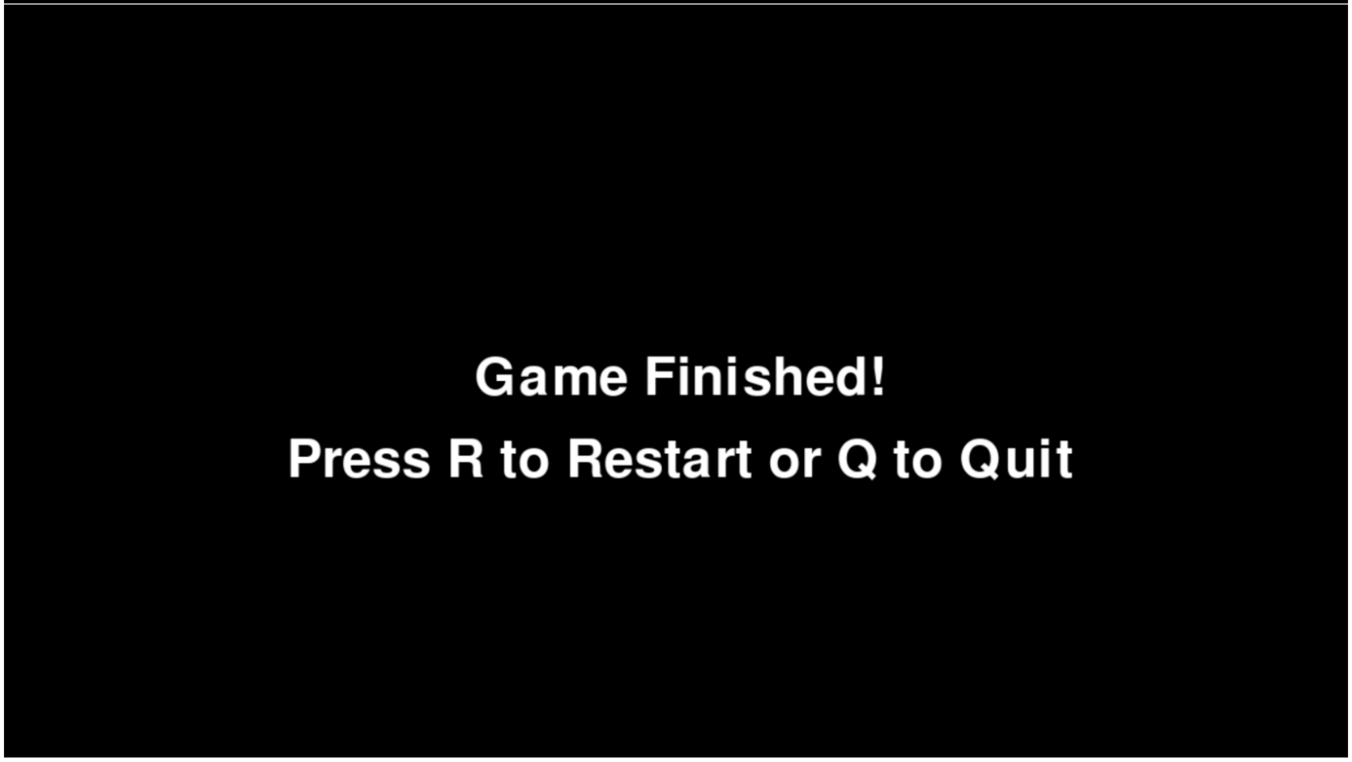
Game Screen



Game Over Screen


Game Over

Press R to Respawn or Q to Quit


Game Finished!

Press R to Restart or Q to Quit

1.2. 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/drive/folders/1datanWn8FotVpL1lKeYpokv5wb21fgnZ?usp=sharing

CHAPTER 4

EVALUATION AND REFLECTION

4.1. Lessons Learnt

What I've learnt through this is that I am REALLY bad at coding :/. I cycled between 3 different projects which I couldn't complete due to its difficulty so I had to go through a lot of trial and error and inspirations from pre existing games and etc. Had to consult some people to get my game fully working and atleast playable.

Took me a decent while to get this game up and running after a bunch of failures in debugging the games I had a moment where I was going to give up in coding a game and make something more simpler, though the result of the game is also quite simple.

What I noticed is that, I really had to have some fun in making these types of stuff so I can always be motivated to improve it and make it better. If it is exciting, it just feels way more rewarding due to the effort being put out and just the pure satisfaction of the game working.

I had underestimated the difficulty of this project which I really shouldn't have as we were given a whole semester to complete this, luckily the time I had left was sufficient enough to complete the game even after the amount of failures I faced.

There were also a bunch of unforeseen things that happened (getting bedridden H-1 of the final project presentation and submission) so a bunch of better aspects have not been improved and some could have been essential parts were not implemented.

4.2. Future Improvements

There are a lot of improvements I could do in the future such as the addition of levels, different types of characters, more enemies, items, power ups and many more. This was all just a matter of time and diligence, with the correct time management this is easily implementable and as long as the motivation is there.

Another thing is to add more maps in between different levels, the game as it stands currently is VERY dull so this addition of more maps and levels will make it more appealing and there will be more of a point to continue in playing the game. A lot of debugging would be required for this and more experimenting with the .tmx file.

Honestly with this project being the very first ever pygame I made, I would say that this overall experience is exciting, with all the excitement and stress taken into account :D. I would like to improve on this game or future projects and this has helped me a lot into improving future projects and how to start them. The way to manage them also comes as a bonus from this experience.

Overall, even though there are still a lot to be implemented, I am pretty satisfied with the result and I hope to do even better in the future!