Algorithm & Programming
Report (L1BC)

Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta

Final Project

World Of Magic

1. Project Specification

1.1 Introduction

The “World Of Magic” is a 2D platform game, players are expected to run throughout the level and get items to meet the required profits. Players may also get boosts that are spread in the game to gain its benefits, and the benefits might help players to reach to the end.

Game Dynamics

- Players are expected to have intermediate skills to jump through the platforms. There are multiple ways to win the game, and players should be able to find their own strategy to win the game.

- There are also some items spreaded in the game, and players should collect them to meet the profit quota, however, there are also some items that have different purposes, such as the sneakers and jetpacks which allows the player to get movement boosts.
- Along the game, players will encounter challenges, the enemies might not be moving, but if you can’t jump through them, you will get hit.

2. Solution Design

2.1 Modules

Player Module

Functionality:

- Handles user input for player movement (left, right) and jumping.
- Manages the player's state, including idle, running, and jumping animations.
- Controls the player's interactions with items and enemies.

Enemy Module:

Functionality:

- Detects collisions with the player, triggering actions such as dealing damage.

Item Module:

Functionality:

- Defines various collectible items with unique effects on the player.
- Manages the interaction between the player and items upon collection.
- Handles the removal of items from the game.

Map and Tile Module:

Functionality:

- Represents the game world through maps composed of tiles.
- Provides collision detection between the player and map elements.

Camera Module:

Functionality:

- Follows the player's movement, ensuring that the camera view adapts to the player's position.
- Provides the 'viewport' of the game world visible to the player.

Scoreboard Module:

Functionality:

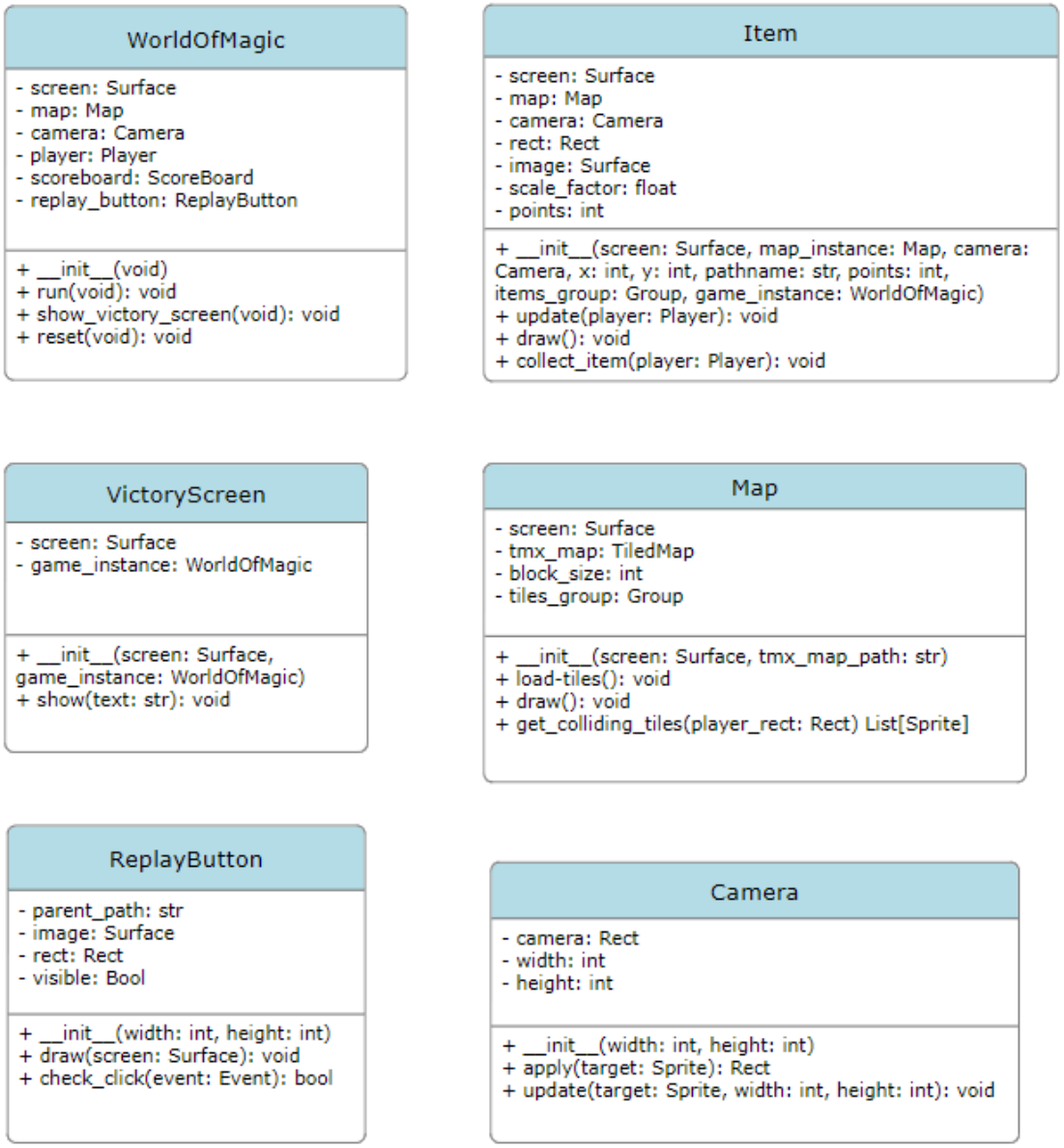
- Displays the player's current score on the screen.
- Updates the score based on collected items or achieved milestones.

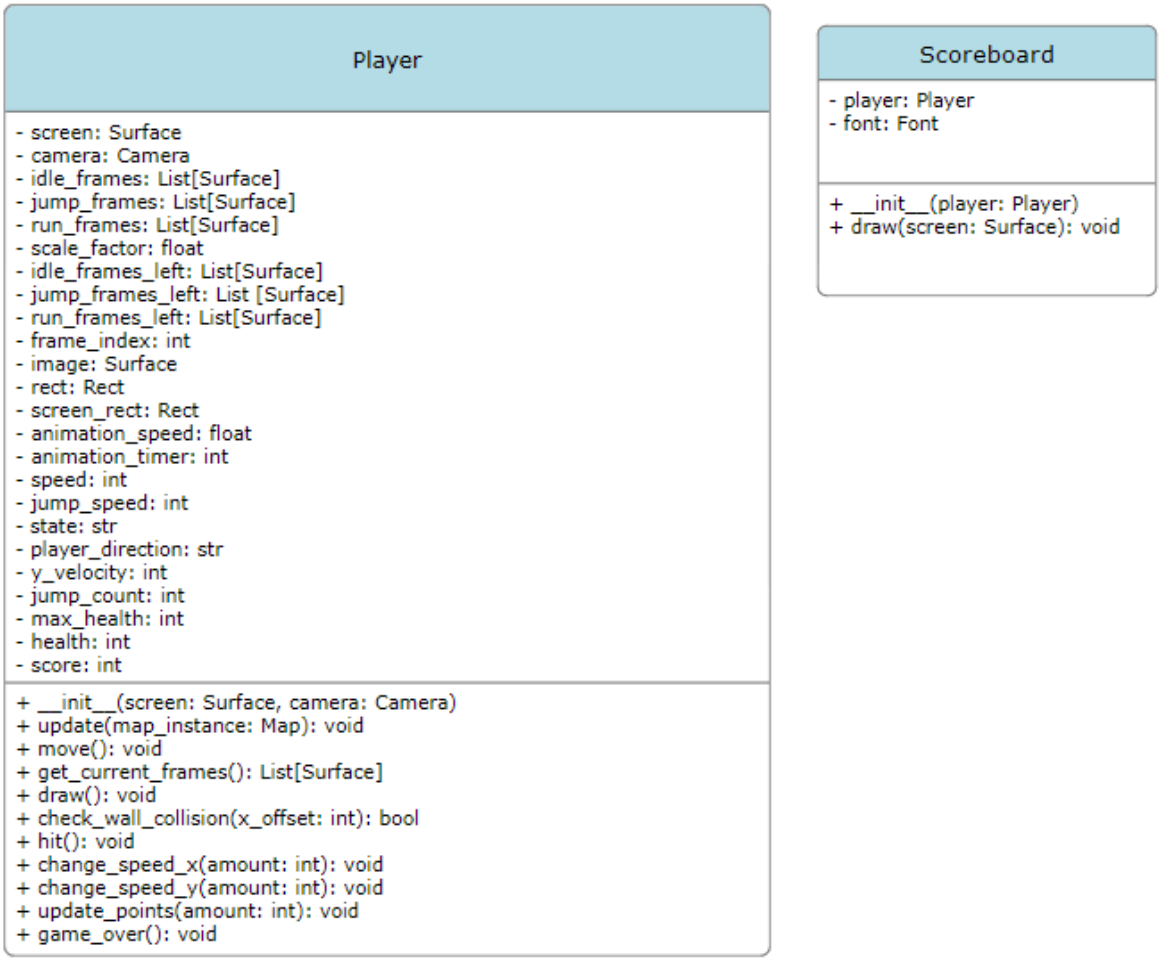
Health Display Module:

Functionality:

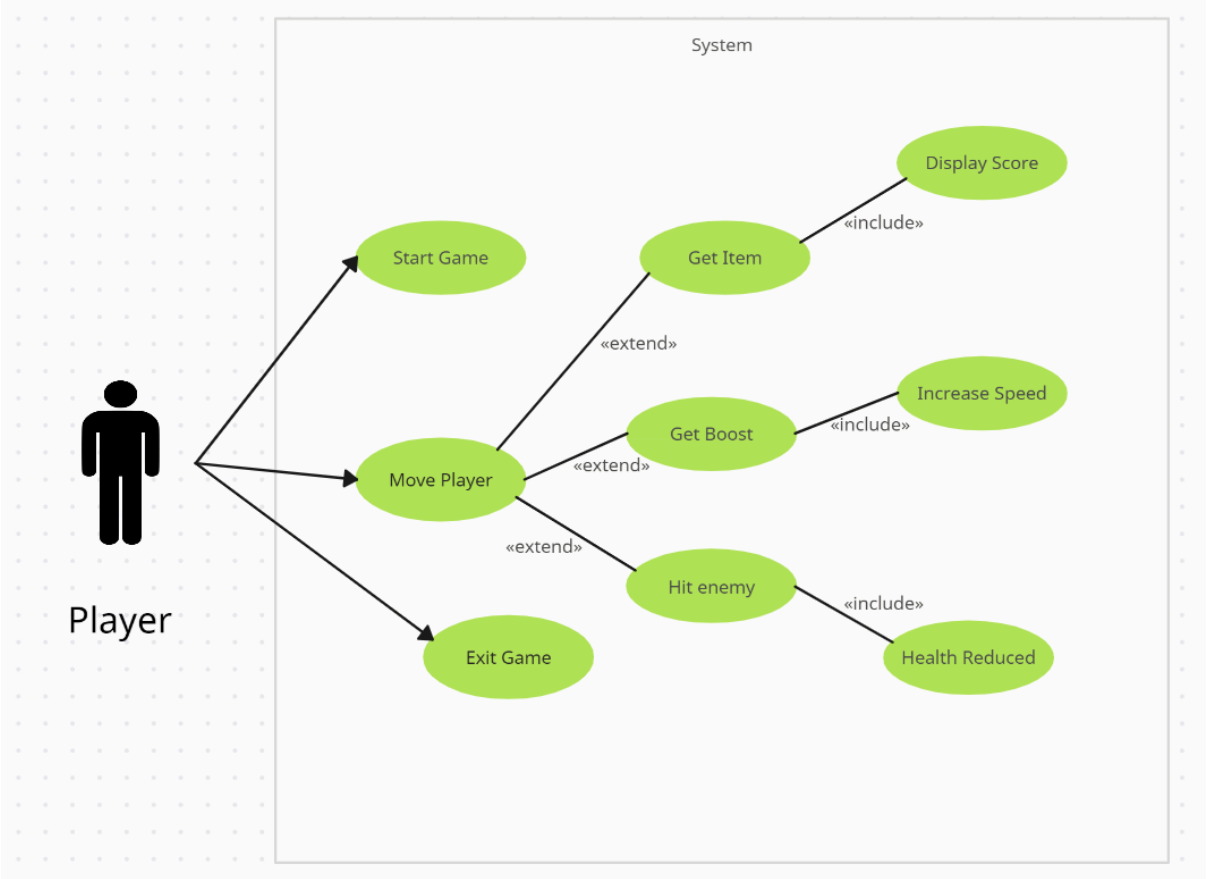
- Visualizes the player's health on the screen.
- Updates the health display in response to enemy collision.

2.2 Class Diagram




Player
- screen: Surface
- camera: Camera
- idle_frames: List[Surface]
- jump_frames: List[Surface]
- run_frames: List[Surface]
- scale_factor: float
- idle_frames_left: List[Surface]
- jump_frames_left: List [Surface]
- run_frames_left: List[Surface]
- frame_index: int
- image: Surface
- rect: Rect
- screen_rect: Rect
- animation_speed: float
- animation_timer: int
- speed: int
- jump_speed: int
- state: str
- player_direction: str
- y_velocity: int
- jump_count: int
- max_health: int
- health: int
- score: int
+ __init__(screen: Surface, camera: Camera)
+ update(map_instance: Map): void
+ move(): void
+ get_current_frames(): List[Surface]
+ draw(): void
+ check_wall_collision(x_offset: int): bool
+ hit(): void
+ change_speed_x(amount: int): void
+ change_speed_y(amount: int): void
+ update_points(amount: int): void
+ game_over(): void
Scoreboard
- player: Player
- font: Font
+ __init__(player: Player)
+ draw(screen: Surface): void

2.3 Use Case Diagram



3. Implementation Details

3.1 Algorithms

Collision Detection Algorithm

I used the pygame collision detection algorithm for the collision between the player, enemies, and also the items.

Functionality:

- The collision between rectangular entities with PyGame allows me to make the player stand on the platform and give a colliding effect for the player when moving on the platform. I also use this as the collision detection when the player collides with the enemies or the items.

Game Physics - Gravity

I also made the basic physics effects, such as gravity to make the game have a realistic falling effect for the player.

Functionality:

- For the player in the game, I created the gravity experience by giving the velocity in y-direction, and removing it when the player collides with the platform. I also implemented the flag for the player to check if the player is jumping or is staying on the platform.

Score Calculation Algorithm

For the game, I also implement the score calculation as the profit quota that the player has to achieve. Players can get the profit quota by collecting items.

Functionality:

- This makes the game have more objectives, players should try to collect the item to achieve the profit quota, and if the player collects an item, then the profit quota displayed will increase until it has reached the required amount.

3.2 Solution Scheme

Object-Oriented Design

I have used Object-Oriented Design for the game, it is done by structuring the game with classes for the entities, and also for the functionalities. This approach makes me enhance code reusability, and make it easier to maintain.

Game Loop

For the game loop, it covers all of the flow of the game. It handles the user input to move the player, and also ensures that all entities are updated throughout the game. It also includes the collision checks between the player and other entities.

Collision Detection

To make the game be realistic, another solution that I used is collision detection. It improves the game by making the player collide with the platform and make the game feel realistic.

Scoring and Game Over

For the scoring, I have 2 ending conditions, where one of them will be achieved if the player reached the profit quota, which should be the only victory case. However, players can also end the game, and it will say that they haven’t met the profit quota. This solution makes the game have different endings, so the player can still explore to get to different endings.

Pygame and Pytmx Integration:

For the game, I used the Pygame library to handle the graphics, input, and other game-related functionalities. This makes the creation of the game easier. Not only that, I also used the Pytmx to load and parse the map. With Pytmx, it makes the level editing to be easier, and instead of using the CSV file and converting it to the tiles one by one, this makes the platform to be done just by having the .tmx file created from Tiled.

3.3 Data Structures

1. **Lists**

   I have also implemented the lists inside the game, and it is used mostly to store the frames for the players and enemies. It makes the animation easier, as I only need to loop on the list.

2. **Rectangles (Pygame Rect Class)**

   I have also utilized the Pygame's Rect class to represent the entity with rectangular regions. This helps all the entities to have a representation as a box, and it causes the collision detection to be easier.

3. **Dictionaries**

   Dictionaries are used for managing key-value pairs of game-related information. Dictionaries are used to get player frames, and therefore it will be easier to return its frame by accessing the data inside the dictionary instead of using if-else statements and writing down what it should return for every player state.

4. **Custom Classes**

   Custom classes are made for this game, such as Player, Enemy, Item, and Scoreboard. These classes are created to encapsulate related data and functionality. It also helps to store the methods required to perform inside the entity.

5. **Group (Pygame Sprite Group)**

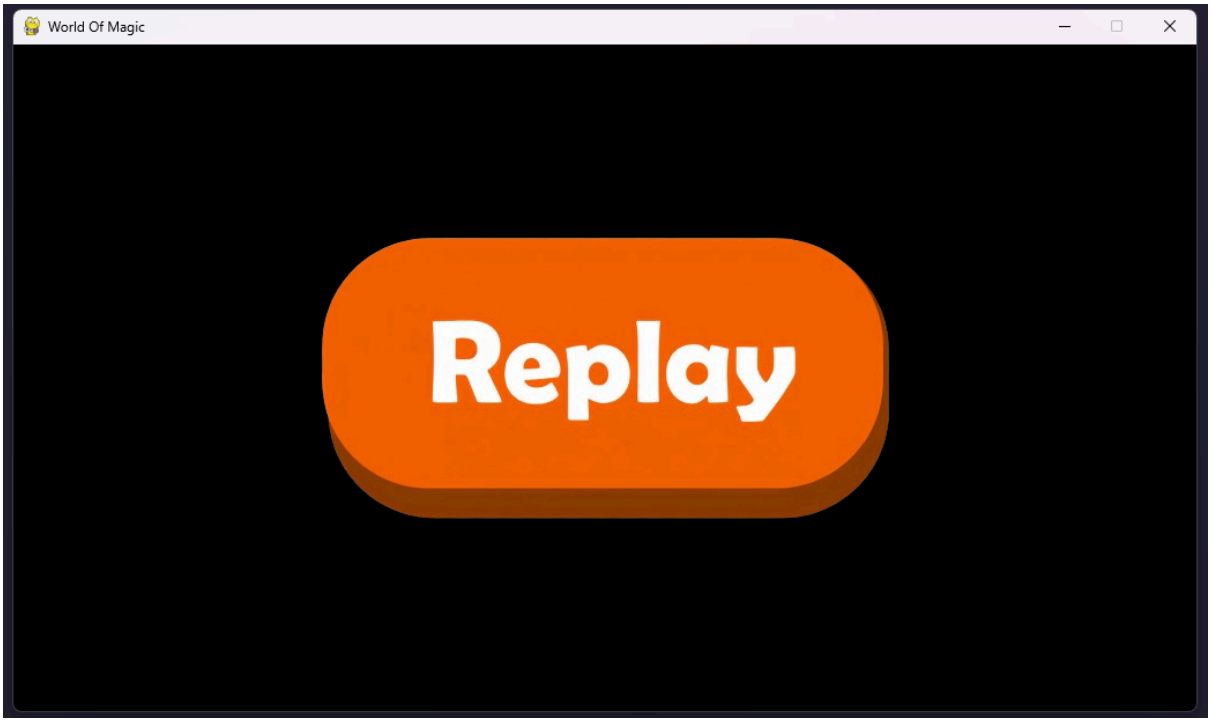
   I have also managed to use the Pygame's Sprite Group to efficiently manage and update multiple sprites at once. This is done for the items, and enemies, and it allows me to make the drawing and update easier as I can loop them by calling the item one by one from the group.

4. Evidence

a. Starting Platform

b. Replay Screen

This screen will pop-up when the player's health is below 0.



c. End Game Screen (When player reach the Profit Quota)

d. End Game Screen (When player doesn't reach the Profit Quota)