OOP Project Final Report

Abdullah Tariq, 29123

May 2025

1 What is the Project?

My project is a GUI-based single player game called "Maze Escape", where the player has to avoid the enemies, collect the coins, collect the key and escape the maze. The player can move in four directions: up, down, left, and right. The player can also engage in a fight if the enemy is in on of the four adjacent cells. Each player has a health, attack, speed and fights which decreases each time the player fights an enemy. It's an inspiration from some games such as **"Pacman", "Bomber Man" and "Dungeon Escape"** from our OOP assignment. The game is played on a grid of cells, where each cell can be empty, contain a coin, an enemy, or the key. The game ends when the player collects the key of the last room or when the player dies. I have also given the choice to the user to choose the player's name and the player itself.

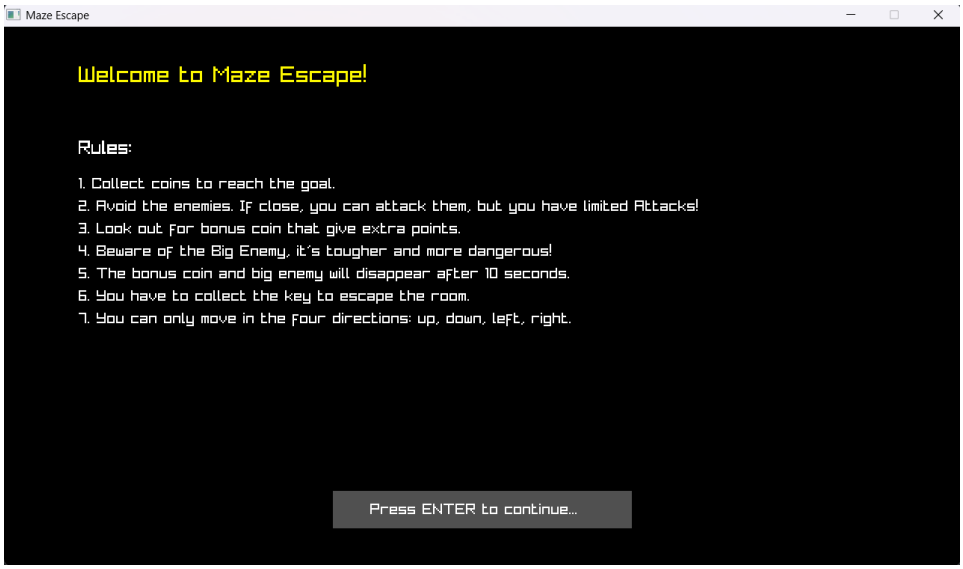


Figure 1: Intro Screen

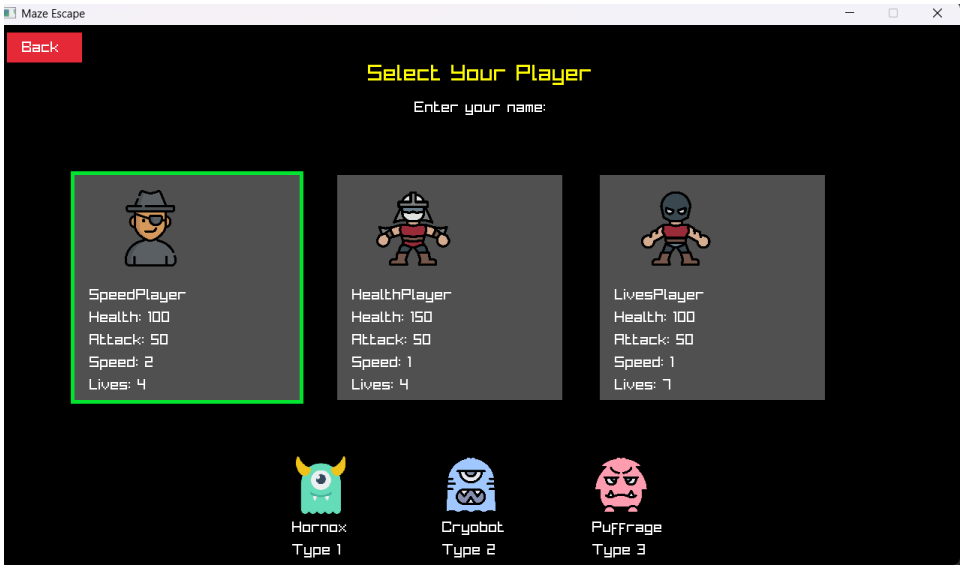


Figure 2: Player Selection Screen

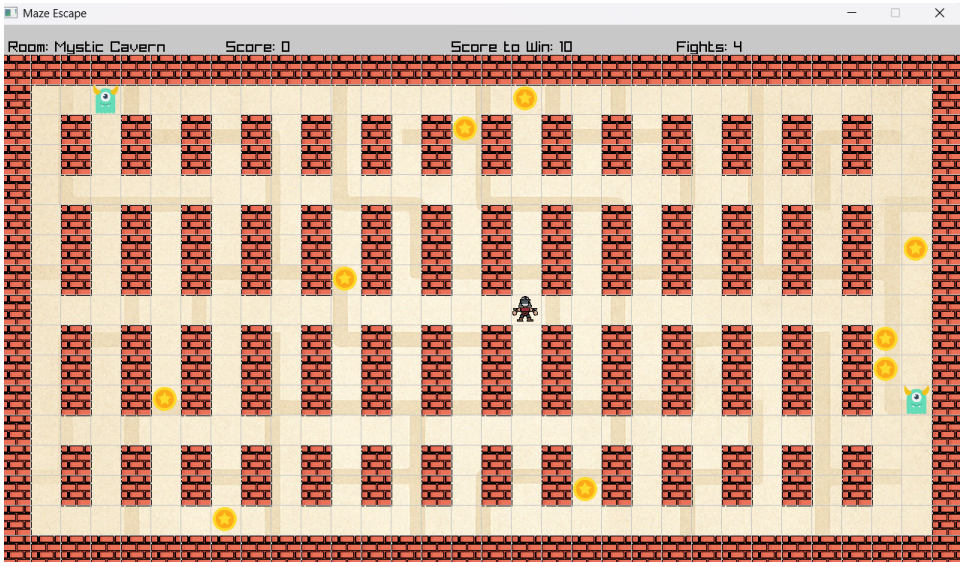


Figure 3: Game Screen

2 How is the Project Implemented?

So first the IntroScreen is displayed, where the user can see the name of the game and the instructions. Then the user is taken to the PlayerSelectionScreen, where the user can select the player type and enter the name of the player. Each screen is returning another upcoming screen, this way it can be used in a while loop where previous screen can be replaced easily and as soon as a nullptr (meaning no screen) is returned, the loop can be exited.

Then the GameScreen is displayed, where the user can play the game. Inside this screen is also a while loop, which is used to keep the game running and running the function which displays the map and every assets on it. The map is a 2D vector of vectors, where each element is a pair of a character and a void pointer. This is done to make the map generic and to be able to use any type of object in it, additionally, each cell being a vector lets me have multiple objects in a single cell which

is useful for the coins and the enemies. The maps are read from a file, where each cell is represented by a character. The characters are then converted to the corresponding objects using a switch case statement.

In the main loop, first the players move function is called, which takes the input from the user and moves the player in the desired direction. Then the enemies move function is called, which moves the enemies in the desired direction. Then the fight function is called, which checks if the player is in a cell adjacent to an enemy and if so, it calls the fight function of the room, which in turns calls the fight function of the player. Then the checkGameStatus function is called, which checks if the player has collected all the coins and the key and if so, it ends the game. In the end if the game is over then a message is displayed and the game is exited if the user presses the exit button otherwise the game is restarted by returning the IntroScreen.

3 How are the OOP Principles Used?

The project is based on the principles of Object-Oriented Programming (OOP), which include encapsulation, inheritance, polymorphism, and abstraction.

3.1 Encapsulation

I have used encapsulation to hide the internal state of the objects and expose only the necessary methods to interact with them. Mostly those methods are getters and setters. For example, the `Player` class has private attributes such as `health`, `attack`, and `speed`, and public methods to get and set these attributes. But some of the attributes such as the position of the player and the enemies are public, since they are accessed by the program a lot of times. It would be a waste of time to create getters and setters for them and plus, they are changed by the program every second.

3.2 Inheritance

I have used inheritance to create a hierarchy of classes that share common attributes and methods. For example, the `Player` class has three subclasses: `HealthPlayer`, `LivesPlayer`, and `SpeedPlayer`, which inherit from the `Player` class and override some of the methods. The `Enemy` class also has three subclasses: `Enemy1`, `Enemy2`, and `BigEnemy`, which inherit from the `Enemy` class and override some of the methods.

3.3 Polymorphism

I have used polymorphism to allow objects of different classes to be treated as objects of a common superclass. For example, as described above, the `Player` class has three subclasses: `HealthPlayer`, `LivesPlayer`, and `SpeedPlayer`, which inherit from the `Player` class and overrides the `move` method. Similarly, the `Enemy` class's subclasses also override their `move` and `findPlayer` methods. This allows me to have separate implementations of the same method for different classes, while still being able to call the method on the base class type.

Another reason I needed to do this was since my maze grid was set up in a way that each element was **void*** so this way I could always point the base class to the void* and then call the method on the base class type. If overridden, it would call the method of the derived class, otherwise it would call the method of the base class as needed.

3.4 Abstraction

I have used abstraction to hide the complex implementation details of the classes and expose only the necessary methods to interact with them. For example, the `Player` and `Enemy` classes, both are abstract classes that define the common attributes and methods for all players and enemies, respectively. They are abstracted because an object should not be created from them directly, but rather from their subclasses.

4 Tool and Libraries

- **IDE:** Visual Studio Code
- **Language:** C++
- **Libraries:**
  - Raylib (For Graphics)
  - vector, unordered_map, string, iostream, cstdlib, pair, algorithm, utility, queue, array, priority_queue, stdexcept, fstream, chrono, thread, and cmath
- **Version Control:** Git and GitHub
- **Documentation:** Latex

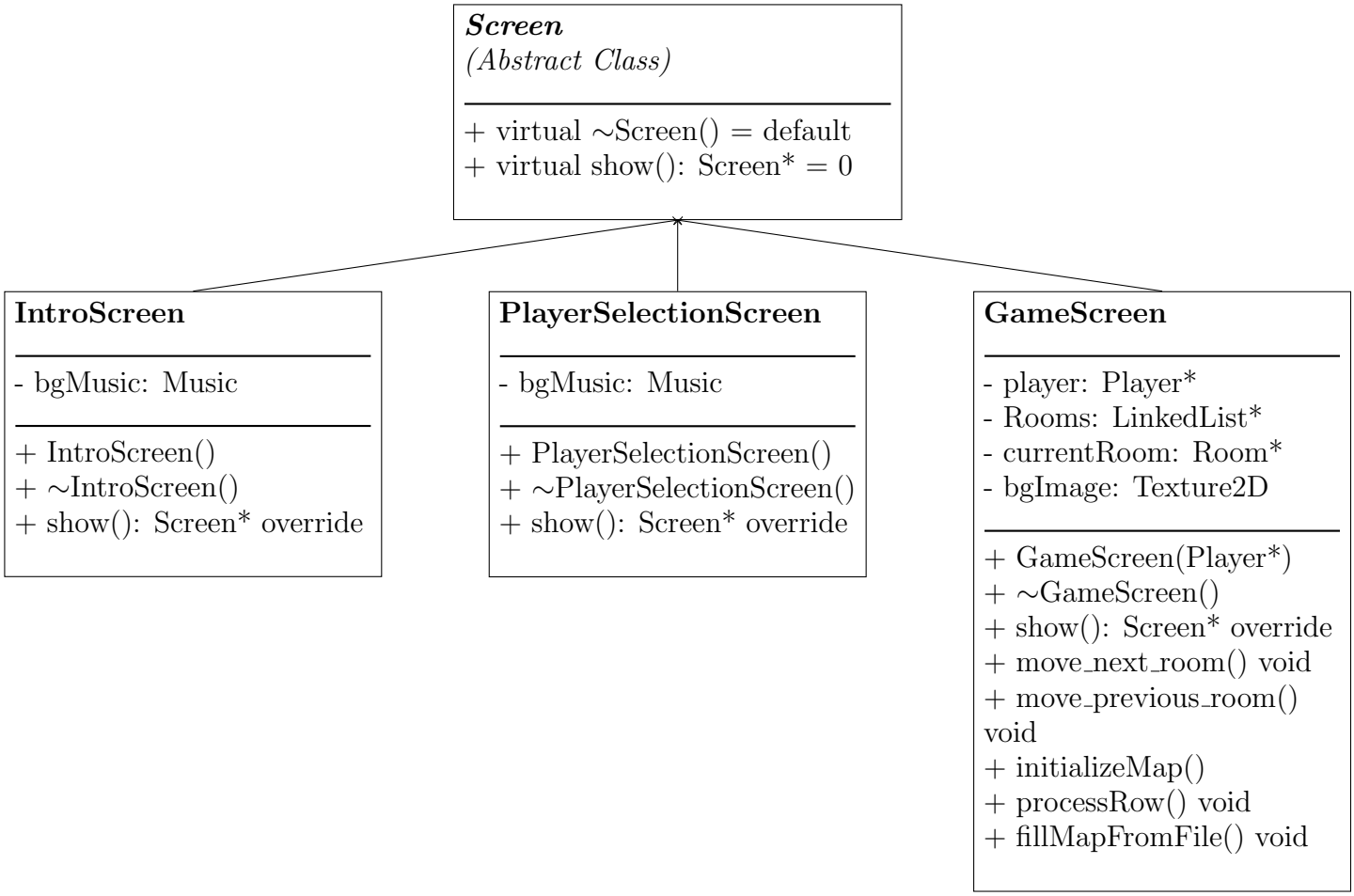
5 TimeLine

- **Feb 21, 2025:** Started ideation and planning of the project.
- **In a week or so:** Created the basic structure of the project and the classes (`.hpp files`).
- **In a month:** Created the whole game and the GUI using Raylib.
- **Till mid of April:** Made modifications, tested the game, and added the features.
- **Till 5th May:** Created the UML diagrams and the documentation.
- **Till 10th May:** Finalized the project and the documentation.

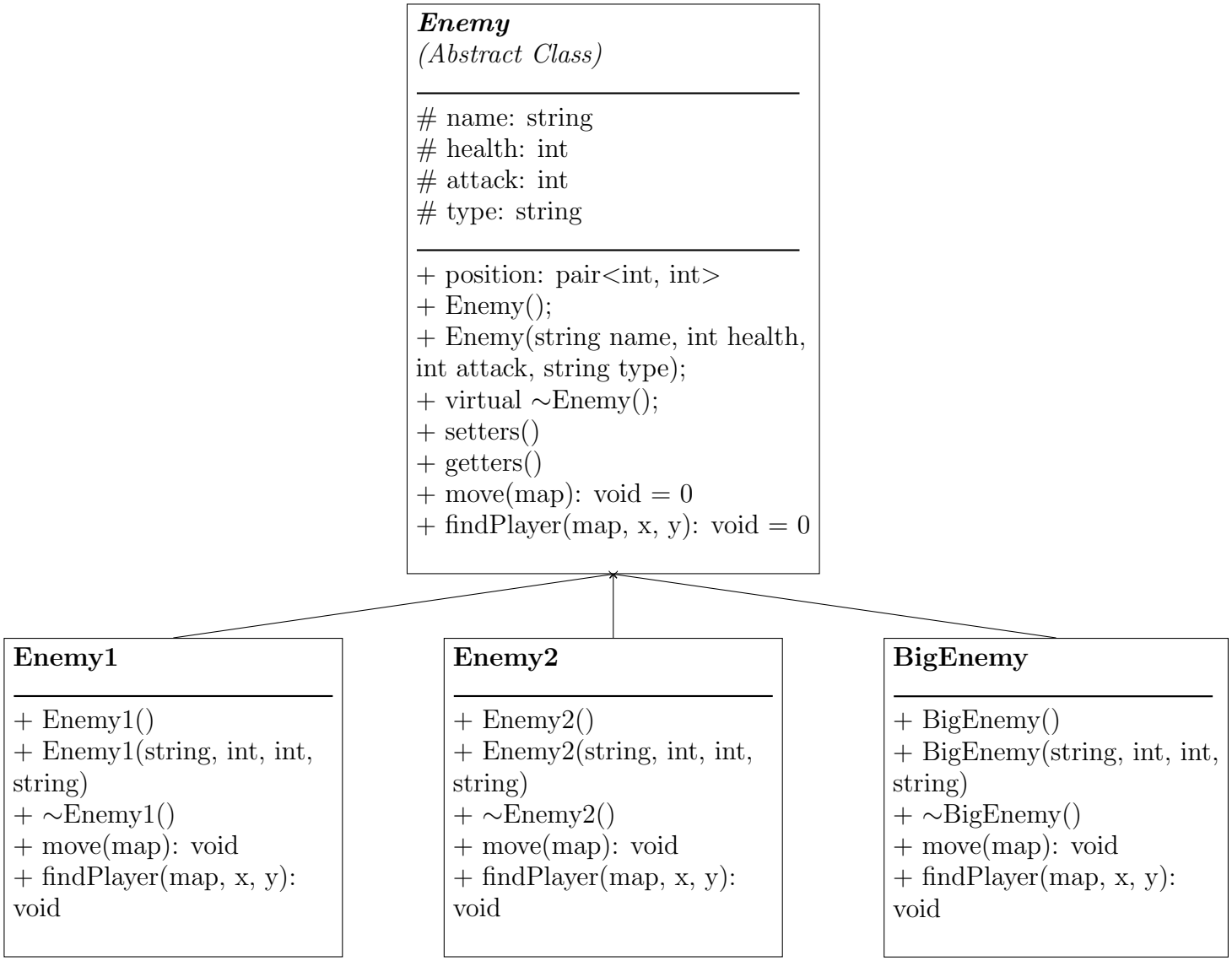
The key milestones were the creation of the classes and the GUI, as they took a lot of time to implement and test. The project was completed in a month, but I had to make some modifications and add some features to it as said in the proposal of the project.

6 UML Diagrams

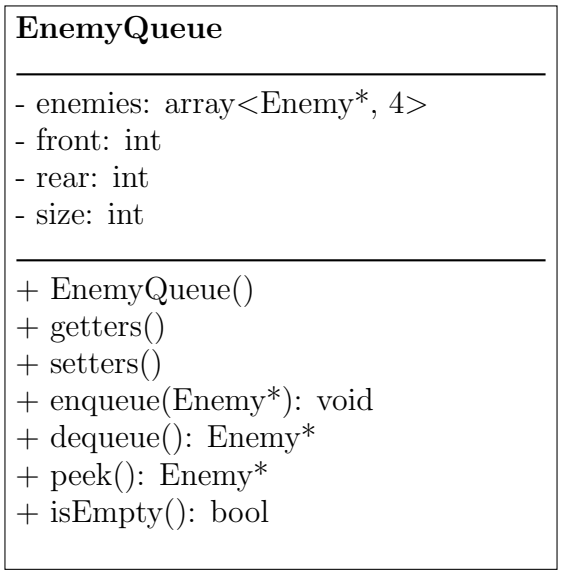
6.1 Screen UML Diagram


Screen
(Abstract Class)
+ virtual ~Screen() = default
+ virtual show(): Screen* = 0
IntroScreen
- bgMusic: Music
+ IntroScreen()
+ ~IntroScreen()
+ show(): Screen* override
PlayerSelectionScreen
- bgMusic: Music
+ PlayerSelectionScreen()
+ ~PlayerSelectionScreen()
+ show(): Screen* override
GameScreen
- player: Player*
- Rooms: LinkedList*
- currentRoom: Room*
- bgImage: Texture2D
+ GameScreen(Player*)
+ ~GameScreen()
+ show(): Screen* override
+ move_next_room() void
+ move_previous_room()
void
+ initializeMap()
+ processRow() void
+ fillMapFromFile() void

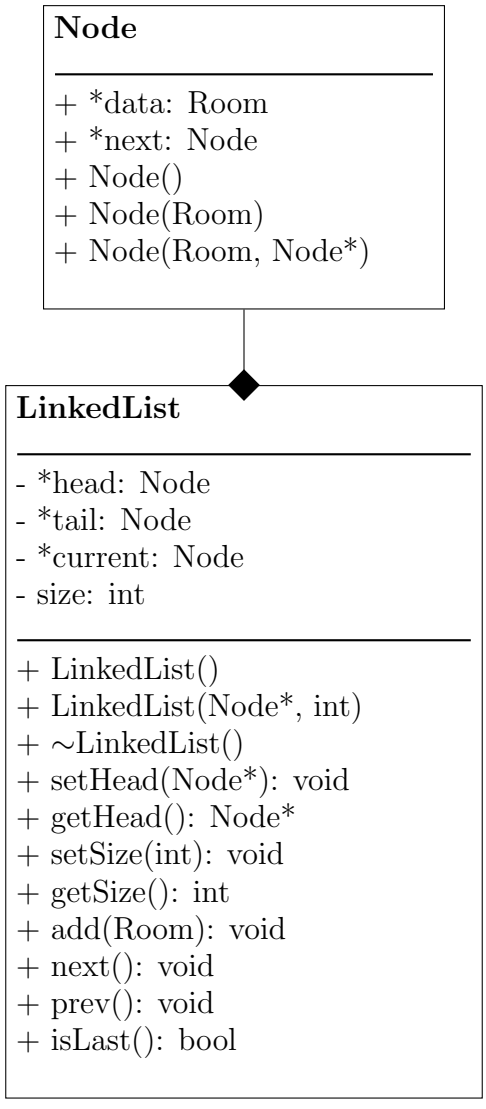
6.2 Enemy UML Diagram



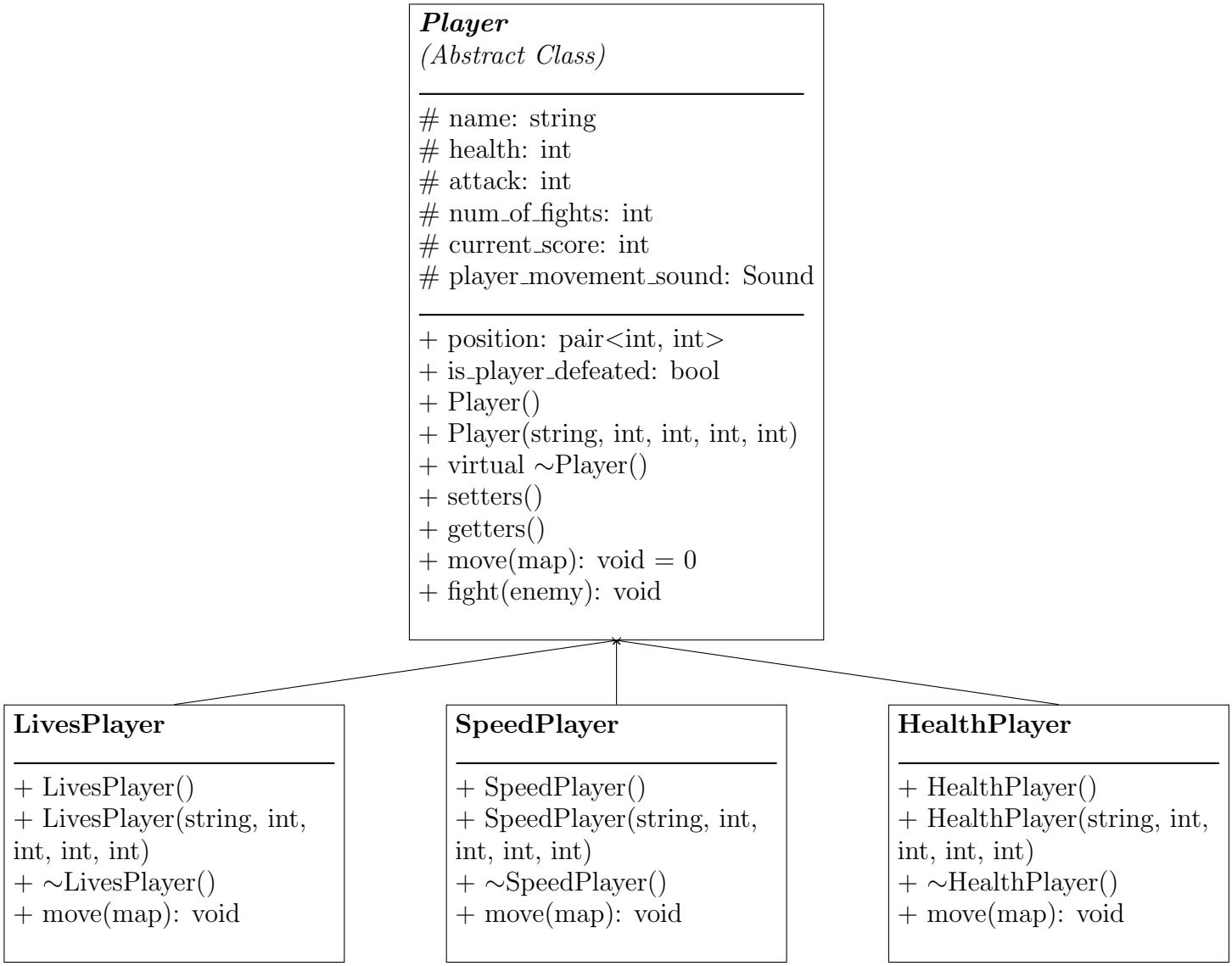
6.3 Enemy Queue UML Diagram



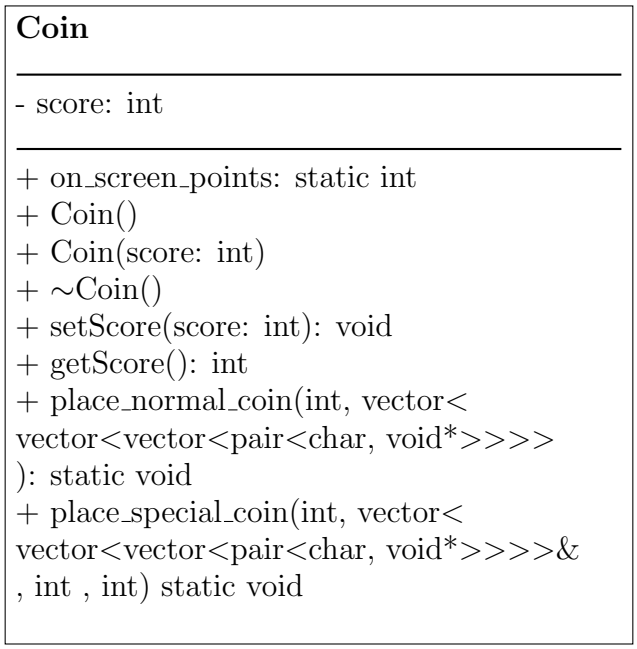
6.4 LinkedList UML Diagram



6.5 Player UML Diagram



6.6 Coin UML Diagram



6.7 Room UML Diagram

Room

- name: string
- map: vector<vector<vector<pair<char, void*>>>>
- enemies: vector<Enemy*>
- num_of_enemies: int
- wallTexture: Texture2D
- LivesplayerTexture: Texture2D
- HealthplayerTexture: Texture2D
- SpeedplayerTexture: Texture2D
- enemy1Texture: Texture2D
- enemy2Texture: Texture2D
- bigEnemyTexture: Texture2D
- coinTexture: Texture2D
- specialCoinTexture: Texture2D
- keyTexture: Texture2D
- enemy_death_sound: Sound
- player_death_sound: Sound

+ is_key_collected: bool
+ is_key_placed: bool
+ score: int
+ score_to_win: int
+ coinPlacedTime: chrono::steady_clock::time_point
+ coinPlaced: bool
+ trackedBigEnemy: Enemy*
+ EnemyQueue: EnemyQueue*
+ Room()
+ Room(map, enemies,
num_of_enemies, score_to_win)
+ ~Room()
+ getters()
+ setters()
+ printMap(Player*): void
+ movePlayer(player*): void
+ moveEnemies(): void
+ fightEnemy(Player*): void
+ checkGameStatus(): bool
+ show_key(vector<vector<vector<pair<char, void*>>>>): void
+ placeCoins(vector<vector<vector<pair<char, void*>>>>&,
int, void (*place_normal_coin)(int, vector<vector<vector<pair<char, void*>>>>&),
void (*place_special_coin)
(int, vector<vector<vector<pair<char, void*>>>>&, int, int), bool): void

7 Testing

For Testing my game, I used Debugging only where I placed *std* :: *cout* statements in the code to check if the values are being passed correctly and if the functions are being called correctly. If at any point the functions or methods would not be called, I would then check those functions and methods to see if they have any errors or not. Other than that, I tried to test each individual function in the main game loop so I would often comment out the other function calls and test one function at a time. My major concern was the memory leaks, so I placed *std* :: *cout* statements in the destructor of each class to see if they are being called or not and sure enough, they were being called.

8 Challenges Faced

Making the map was a challenge for me, since I was reading the map from a .txt file and in that I had placed characters like `'.', 'C', '1', '2', 'K', 'P'` to represent the entities in the game, which were in the file really close to each other. But when I read the file and the GUI appeared, they were much bigger since now there were images, not characters. A problem linked to this was that since the spacing of the characters was too close in the text file I after some days realized that I had the number of rows and columns wrong in some cases.

Making the pathfinding algorithm was also a challenge for the enemies, since I had to make sure that the enemies would not go through the walls and would not go out of the map. Other conditions were that the enemies effectively find the player and not get stuck in a loop or not do nothing at all. This happened at first and for testing it, I printed out the coordinates of player and the enemies but still no solution was found. It was only after a few days when my map was shown in the GUI that I realized my player was surrounded by walls and the enemies were not able to find the player hence they were not moving at all. I also had to add some percentage errors to the enemies so they would not always find the player and would not always move towards the player. This brought some randomness to the game and made it more fun to play.

The last challenge that I feel like is for everyone is that to decide which functions are needed and in which class it should go to. Over time I made changes to the classes and added some functions to the classes which I thought were needed, only this way I was able to make the classes correctly and make the game work.

9 Compilation

To compile the project, I used the following command in the terminal:

```
g++ -o maze.exe Sources/*.cpp  Main/main.cpp -I C:\raylib1\include
-L C:\raylib1\lib -lraylib -lopengl32 -lgdi32 -lwinmm
```

Lets give a breakdown of the command:

- `g++` is the compiler used to compile the C++ code.
- `-o maze.exe` specifies the output file name.
- `Sources/*.cpp` includes all the source files in the Sources folder.
- `Main/main.cpp` includes the main file of the project.

- `-I C:\raylib1\include` specifies the path to the Raylib header files.
- `-L C:\raylib1\lib` specifies the path to the Raylib library files.
- `-lraylib` links the Raylib library to the project.
- `-lopengl32 -lgdi32 -lwinmm` links other required libraries for Windows.

After running the command, the executable file `maze.exe` is created in the same directory as the source files. To run the game, simply execute the `maze.exe` file:

```
.\maze.exe
```

I have made another alternate method to run the game, which is using the *.vcode* folder where I have placed the `launch.json` file. This file contains the configuration to run the game using Visual Studio Code. I have placed the commands to compile the game and then run the game in the `launch.json` file. All the user has to do is to run the main file using the green play button in the top right corner of the Visual Studio Code window. This will automatically compile the game and run it.

10 GitHub Repository

The GitHub repository for the project can be found at the following link:

```
https://github.com/Abdullahprogramme/OOP-Project
```

The repository contains all the source files, header files, and the main file of the project. Along with the `.vscode` folder, and other files I made for the project. The repository is public and can be accessed by anyone.