**INTERNATIONAL UNIVERSITY**

**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**

**School of Computer Science and Engineering**

**-----\*\*\*-----**

Ảnh có chứa vòng tròn, biểu tượng, văn bản, Nhãn hiệu

Mô tả được tạo tự động

**PROJECT REPORT**

**SAVING SIR. NGHIA**

**Advisor: Dr Tran Thanh Tung and Nguyen Trung Nghia**

**Course: Object – Oriented Programming**

|  |  |  |
| --- | --- | --- |
| No. | Full Name | Student’s ID |
| 1 | Đỗ Thanh Bảo Anh | ITDSIU22175 |
| 2 | Phạm Tuấn Đăng Khoa | ITITIU22087 |
| 3 | Nguyễn Minh Kha | ITITIU22079 |
| 4 | Trần Thiên Phú | ITITIU22124 |

**TABLE OF CONTENTS**

[ABSTRACT 2](#_Toc20916)

[CHAPTER 1: INTRODUCTION 3](#_Toc13278)

[1. Objectives 3](#_Toc2636)

[2. The Tools Used 4](#_Toc13102)

[3. Developer Team 4](#_Toc2869)

[CHAPTER 2: SYSTERM DESIGN 5](#_Toc17804)

[1. Package Structure 5](#_Toc24561)

[2. List of Class and Responsibility 5](#_Toc17012)

[3. UML Diagram 10](#_Toc7179)

[4. Code Flow 15](#_Toc2388)

[CHAPTER 3: ENTITY, NPCs, OTHER OBJECT DESIGN 25](#_Toc6910)

[1. Player 25](#_Toc16992)

[2. Enemy 25](#_Toc15434)

[3. NPCs 26](#_Toc11978)

[4. Other Object 26](#_Toc25925)

[CHAPTER 4: MAP DESIGN 28](#_Toc19408)

[1. Asset 28](#_Toc7789)

[2. How To Design 29](#_Toc26442)

[3. How To Load 29](#_Toc4162)

[CHAPTER 5: GAME DESIGN 30](#_Toc31135)

[1. Game Rule 30](#_Toc1006)

[2. Game Structure 30](#_Toc30781)

[3. Game Play 30](#_Toc28238)

[4. Animation States 31](#_Toc11537)

[4.1. Player: 31](#_Toc20)

[4.2. Player Attack: 32](#_Toc17047)

[4.3. Monster: 32](#_Toc19478)

[4.4. Monster Attack: 33](#_Toc29041)

[5. UI 34](#_Toc23821)

[6. Audio 36](#_Toc25508)

[CHAPTER 6: DEMO 36](#_Toc7756)

[CHAPTER 7: CONCLUSION AND FUTURE WORKS 37](#_Toc17144)

[1. Conclusion 37](#_Toc22332)

[2. Future works 37](#_Toc21253)

[3. Acknowledgment 37](#_Toc14236)

[REFERENCES 38](#_Toc15601)

**ABSTRACT**

This report focuses on the development of a 2D Top-Down RPG desktop game, "Saving Sir. Nghia." Inspired by Soul Knight [1], the project features various dungeons for players to explore. It incorporates a player experience system with three distinct character skins and intelligent, computer-controlled enemies – both melee and skill-based – to challenge players. Unlike traditional Top-Down RPGs where players attack with swords, this game emphasizes searching for keys and unlocking doors. "Saving Sir. Nghia" explores a new dimension in traditional RPGs by seamlessly blending puzzle-solving with survival elements, where players must defeat enemies to survive. With its simplicity, the game aims to recapture the fun of classic titles like Roguelike, Pokémon, and Mario, while incorporating modern features.

Keywords: dungeon, top down, adventurous, game 2D, object-oriented programming.

**CHAPTER 1: INTRODUCTION**

1. **Objectives**

The goal of the project is to create a 2D game based on top-down game concepts, knowledge of the Java programming language, and basic available design patterns. The game also demonstrated the strengths of object-oriented programming methods in creating games. This is a quite classic game with simple steps, does not require high skills or complexity in the gameplay but still brings an attractive feeling, making players easily attracted to the rhythm. game level. Saving Sir. Nghia allows us to experience the feeling of the player himself as a companion of the main character, accompanying the character through dangerous areas, fighting with monsters, and It's time to find a way to unlock dangerous organs to free yourself. With the efforts of our team members, we have created a program that can provide users with extremely enjoyable experiences. Additionally, we will also try to fix and add more features in the future.

To put it more succinctly, the purpose of the project is

* Helps us better understand Game Development Concepts such as game loops, collision detection, rendering, and game physics...
* Apply knowledge of Java programming language and Object-Oriented Programming to create a game that can bring attraction and entertainment to players.
* Experience the processes of ideating, creating, managing, and improving games.

1. **The Tools Used**

* IDE for programming and debugging: IntelliJ, VSCode, Eclipse.
* Design: Piskel, Simple2DTileEditor.
* Java Development Kit: 21.
* Mean of code version management: GitHub.
* Means of contacting: Facebook

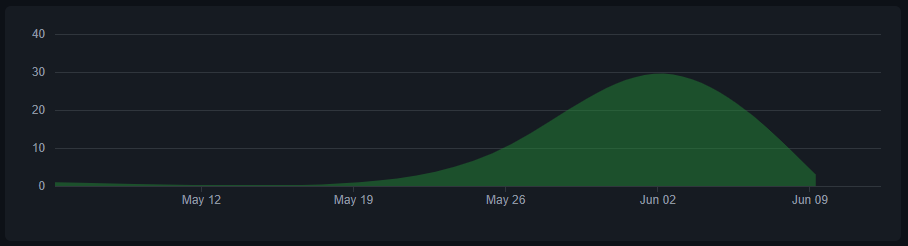


Figure 1: GitHub statistics

1. **Developer Team**

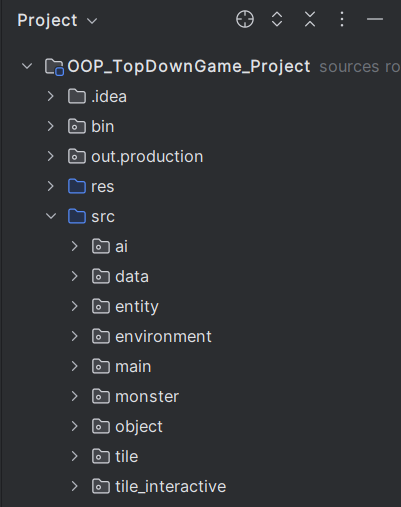
|  |  |
| --- | --- |
| **Name** | **Contribute** |
| Đỗ Thanh Bảo Anh | * Write Report, Slide, Presentation * Develop Code: Class Boss, Screen Level Up,... * Fix Bug |
| Phạm Tuấn Đăng Khoa | * Write Report, Slide, Presentation * Design Map * Develop Code * Fix Bug |
| Nguyễn Minh Kha | * Write Report, Slide, Presentation * Find Asset * Design Player Character, NPC,... * Develop Code: UI, Key System,.. |
| Trần Thiên Phú | * Write Report, Slide, Presentation * Develop Code: UI, Torch,... * Fix Bug |

**CHAPTER 2: SYSTERM DESIGN**

1. **Package Structure**

The project's current structure is the result of extensive iteration and troubleshooting to arrive at this optimized configuration, as the carefully crafted project layout enables the seamless execution of the game algorithms powering our latest creation and driving the interactive experiences in our game.

To facilitate the management of the classes, we have organized them into distinct groups, like:



*Figure 2: Package Structure*

Here is the link to the GitHub repository for our project: https://github.com/Khoataphat/OOP\_TopDownGame\_Project

1. **List of Class and Responsibility**

|  |  |  |
| --- | --- | --- |
| Package | Class Name | Responsibility |
| ai | Node | provides functionality to represent and manage individual nodes in a grid-based pathfinding algorithm. |
| PathFinder | is responsible for finding a path between two points in the game world. It uses a grid-based pathfinding algorithm to navigate through the game's tiles and obstacles. |
| data | Progress | store the progress of the game in a static variable called skeletonLordDefeated. |
| entity | Entity | serves as a representation of various entities within a game world, such as characters, NPCs (non-playable characters), or objects. It encapsulates the necessary features and properties to accurately portray these entities in the game. |
| NPC\_Merchant | which extends the "Entity" class, represents a merchant NPC (non-playable character) in the game. This class is responsible for defining the behavior and attributes of a merchant NPC, including their dialogue options, inventory of items, and interaction functionality, enriching the game world with trade opportunities. |
| NPC\_OldMan | which extends the "Entity" class, represents an instructor NPC (non-playable character) in the game. This class is specifically designed to define the behavior and attributes of an instructor NPC, providing necessary functionalities to provide helpful guidance to players. |
| Particle | which extends the "Entity" class, provides the necessary functionalities to create and manage particles in the game, allowing for dynamic visual effects such as sparks. |
| Player | which extends the "Entity" class, is the embodiment of the hero in our grand adventure.  It encapsulates the behavior and attributes of the player, providing various functionalities to handle movement, interaction, and other actions. |
| PlayerDummy | which extends the "Entity" class, represents a dummy player entity in the game. It serves as a placeholder or test entity during development and does not represent the actual player-controlled character. |
| Projectile | extends the "Entity" class, representing a projectile entity in the game which can be used for ranged attacks or as objects that can move through the game world. |
| environment | Lighting | is responsible for managing the lighting effects. |
| main | AssetSetter | implementing an asset setter for a game. It allows the user to set specific positions for initializing monsters within the game world. |
| CollisionChecker | responsible for detecting and handling collisions between entities, objects, and the player. |
| Config | is designed to manage the game's configuration settings, such as fullscreen mode and volume levels for music and sound effects. |
| CutsceneManager | is responsible for managing cutscenes in our game. It handles the sequence of events, animations, and transitions that occur during a cutscene. |
| EntityGenerator | is responsible for generating different types of Entity objects based on the given itemName. |
| EvenHandler | handles different events and interactions in the game, such as set Dialogue, speak, and hit. |
| EvenRect | initialize a rectangular region in the game to define the boundary of the event region to support collision detection with the player's solid region. |
| GamePanel | inherits from the JPanel class and implements the Runnable interface, is responsible for managing and displaying the game interface. It handles game states, updates components, and renders game elements on the screen. |
| KeyHandler | handles key events in our game. It implements the KeyListener interface, which allows it to listen for and respond to key presses, releases, and typed events. |
| Main | sets up the main game window, configures its properties, and starts the game loop to begin gameplay. |
| Sound | handle sound effects in our game. It provides methods for loading and playing various sound files. |
| UI | responsible for handling various graphical user interface elements and interactions in the game. |
| Utility Tool | responsible for scaling images in our game, allowing for easy resizing and manipulation of images for various purposes. |
| monster | MON\_Bat | extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "Bat" in the game. |
| MON\_GreenSlime | extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "GreenSlime" in the game. |
| MON\_Orc | extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "Orc" in the game. |
| MON\_RedSLime | extends the "Entity" class,responsible for defining the behavior and attributes of a specific type of monster called "RedSlime" in the game. |
| MON\_SkeletonLord | Extends the “Entity” class, responsible for defining the behavior and attributes of the final boss. |
| object | OBJ\_Chest | represents a treasure chest in your game, handling contents with player interaction, and its open/closed state. |
| OBJ\_Coin\_Bronze | represents a bronze coin in your game, handling its value, price, and the action of using it to increase the player's coin count. |
| OBJ\_Door | represents a locked door in your game, requiring a key to open and displaying a message to the player. |
| OBJ\_Door\_Iron | represents an unbreakable iron door in your game, displaying a message that it cannot be opened. |
| OBJ\_Fireball | represents a fireball projectile of player. |
| OBJ\_Heart | represents a heart item in your game, handling its value and the action of using it to increase the player's health. |
| OBJ\_Key | represents a key item in your game, handling its use to open doors and displaying messages accordingly. |
| OBJ\_Lantern | class represents a lantern in game. |
| OBJ\_ManaCrystal | represents a mana crystal item in your game, handling its value and the action of using it to increase the player's mana. |
| OBJ\_Potion\_Red | represents a red potion in your game, handling its use to heal the player and displaying a message confirming the health increase. |
| OBJ\_Rock | represents a rock projectile of monster. |
| OBJ\_Sword\_Normal | represents a normal sword in your game, defining its properties like attack power, attack area, and animation durations. |
| OBJ\_thay\_nghia | represents an item related to the character "Teacher Nghia". When interacted with, it triggers the ending scene. |
| tile | Map | provides functionality for creating, managing, and displaying the game map in both full screen and mini map modes. |
| Tile | represents individual tiles with their visual appearance and collision status |
| TileManage | manages the tiles used in the game, including loading tile data, setting up tiles, loading map data, and drawing the tiles on the game panel. |
| tile\_interactive | Interactive | is a subclass of the "Entity" class and represents interactive tiles in the game world. |
| IT\_AreaAttack | is a subclass of the "InteractiveTile" class and represents an interactive tile called "Area Attack" in the game world. |

*Table 1: List of class and responsibility*

1. **UML Diagram**

We provided the UML diagrams for the entire project and each group that was addressed to help you better understand the structure and methods. Whole UML Diagram: https://github.com/Khoataphat/OOP\_TopDownGame\_Project/blob/91b688f75caeb4fb7a783d08951a4c38c9cef223/UML\_OOP\_Project.pdf

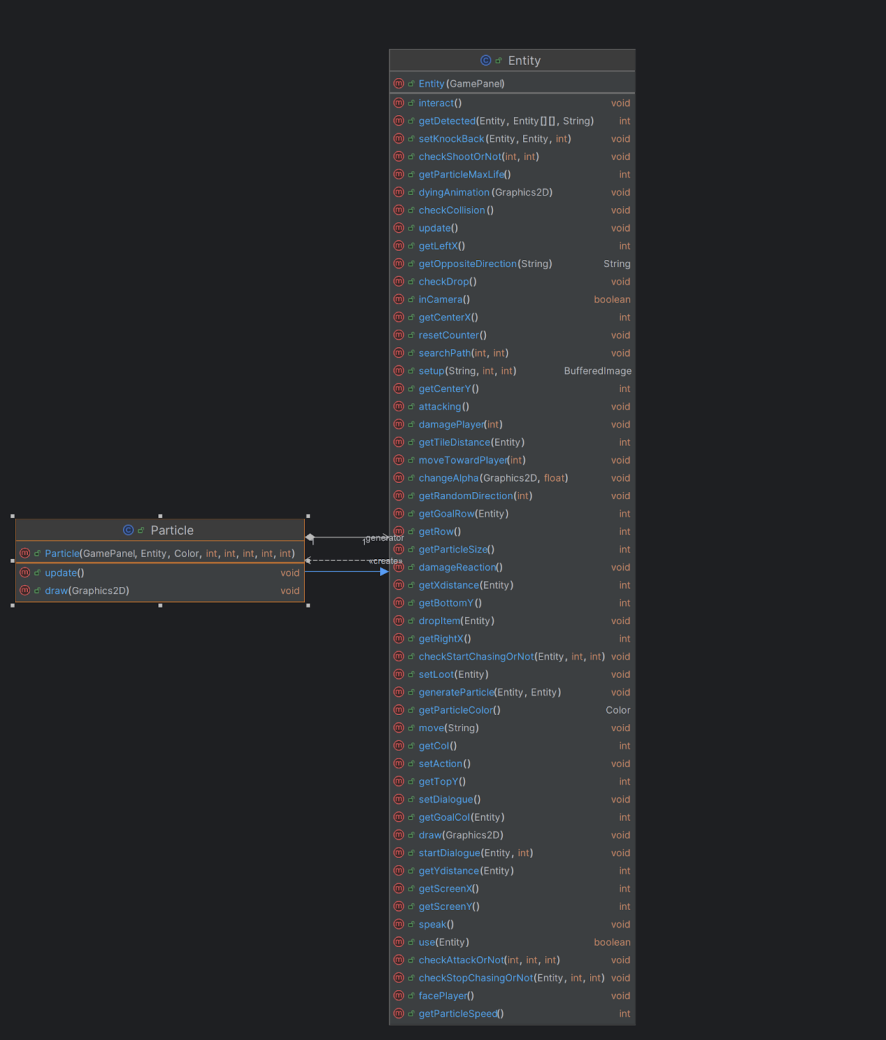


Figure 3: Entity diagram

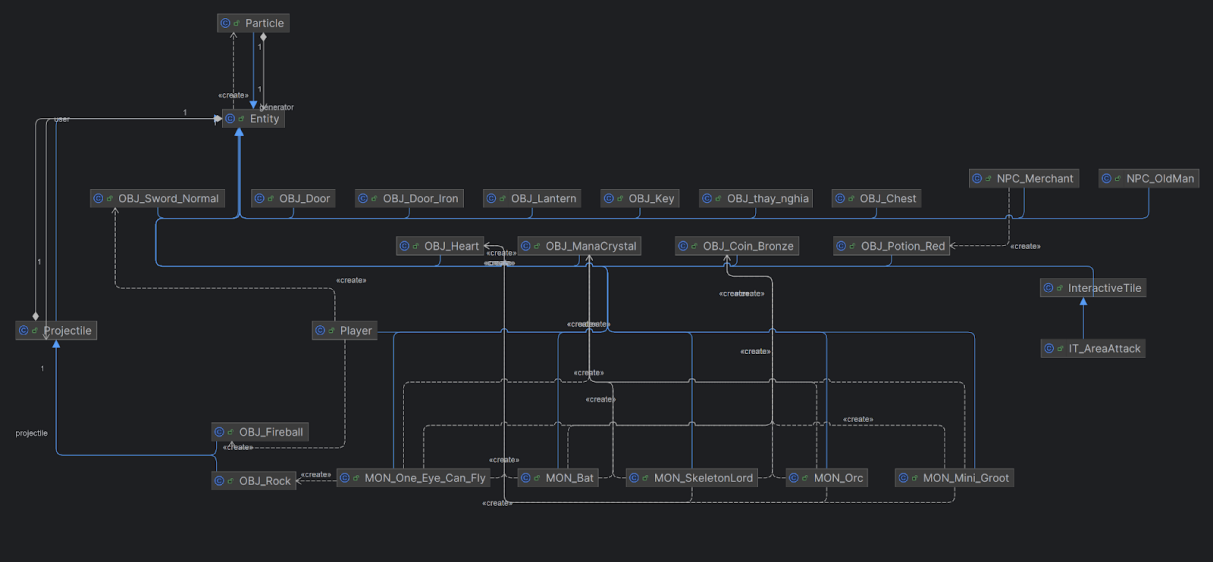


Figure 4: Class extend from Entity diagram



Figure 5: Player diagram

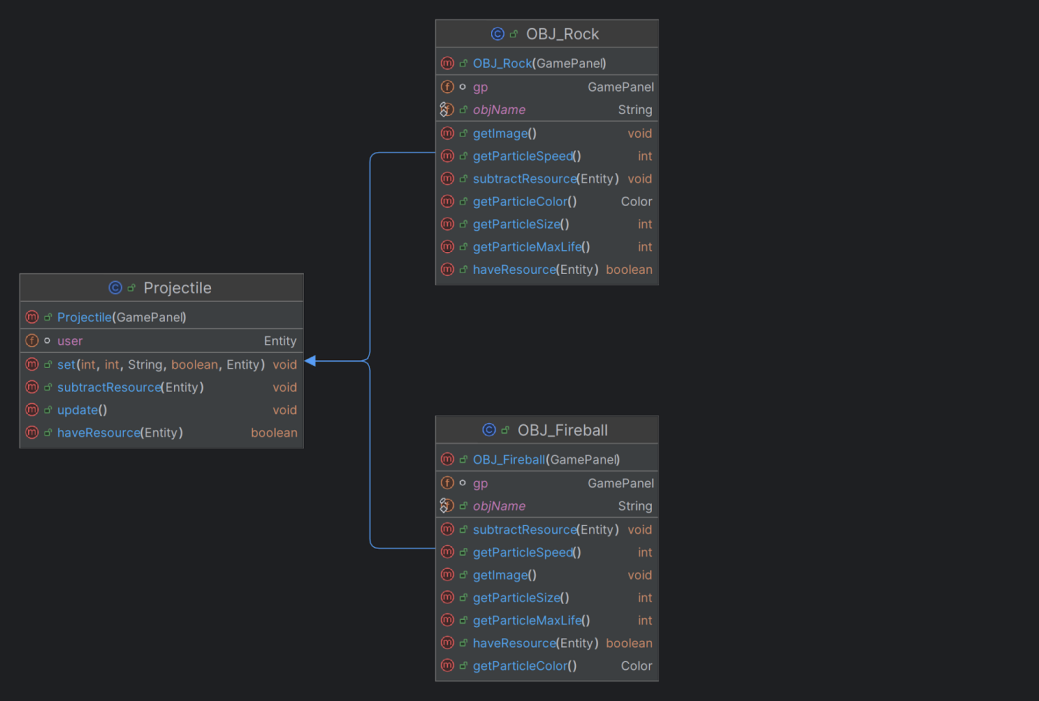


Figure 6: Projectile diagram

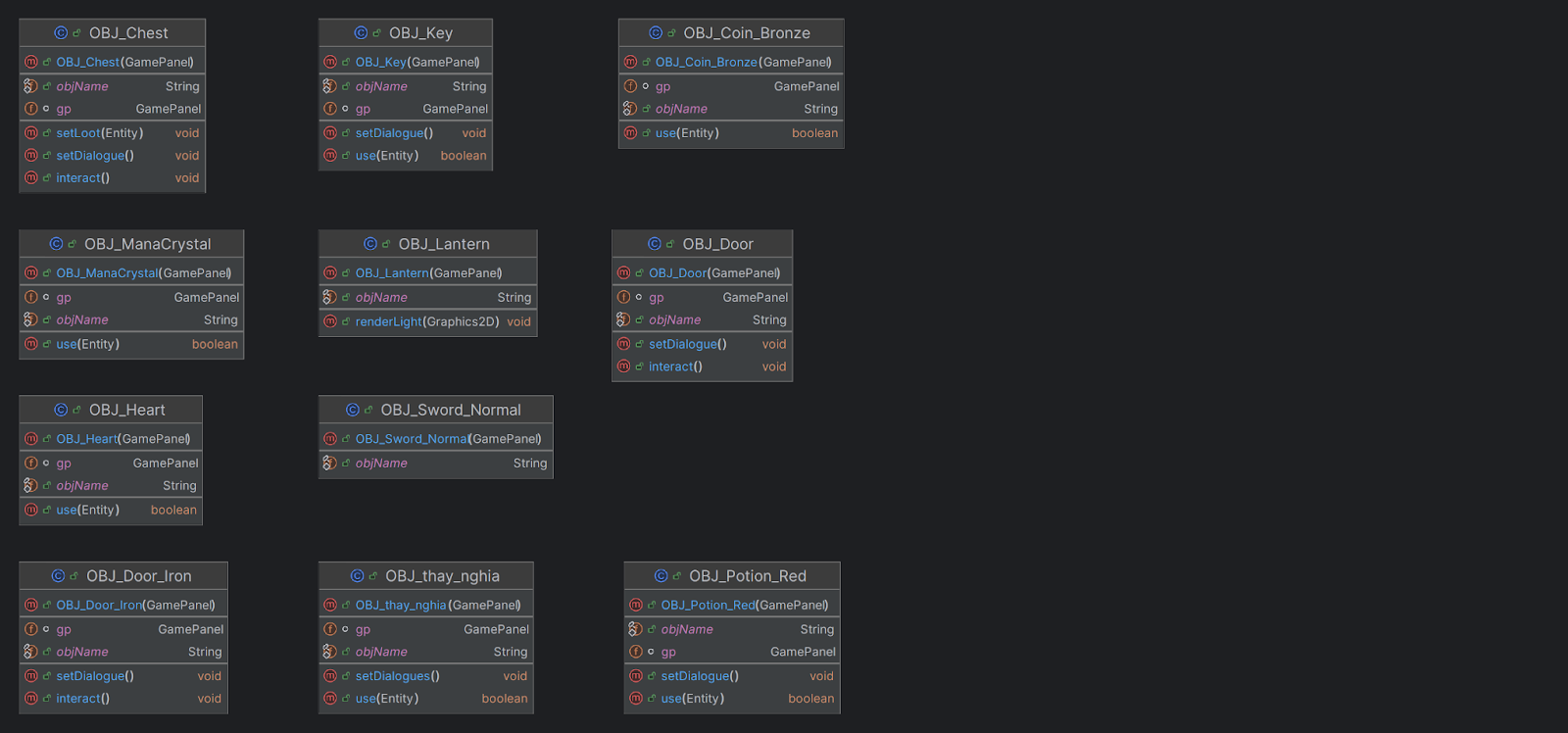


Figure 7: Object diagram

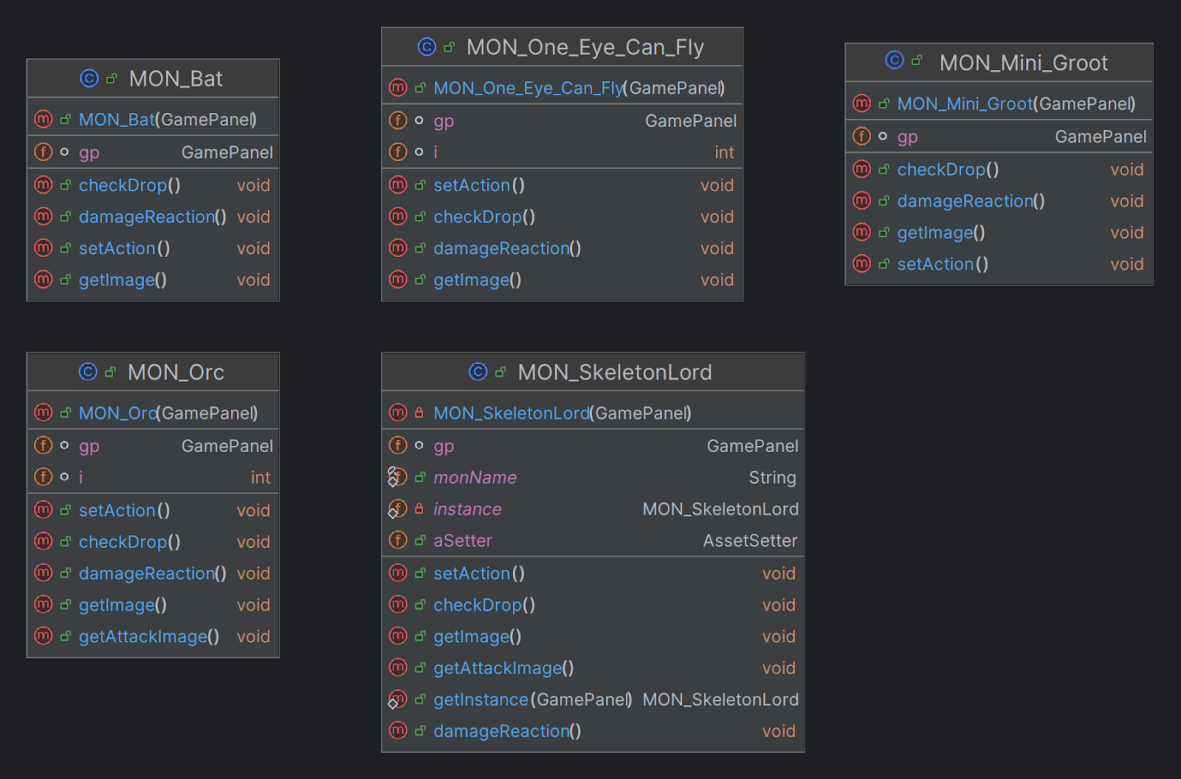


Figure 8: Monster diagram

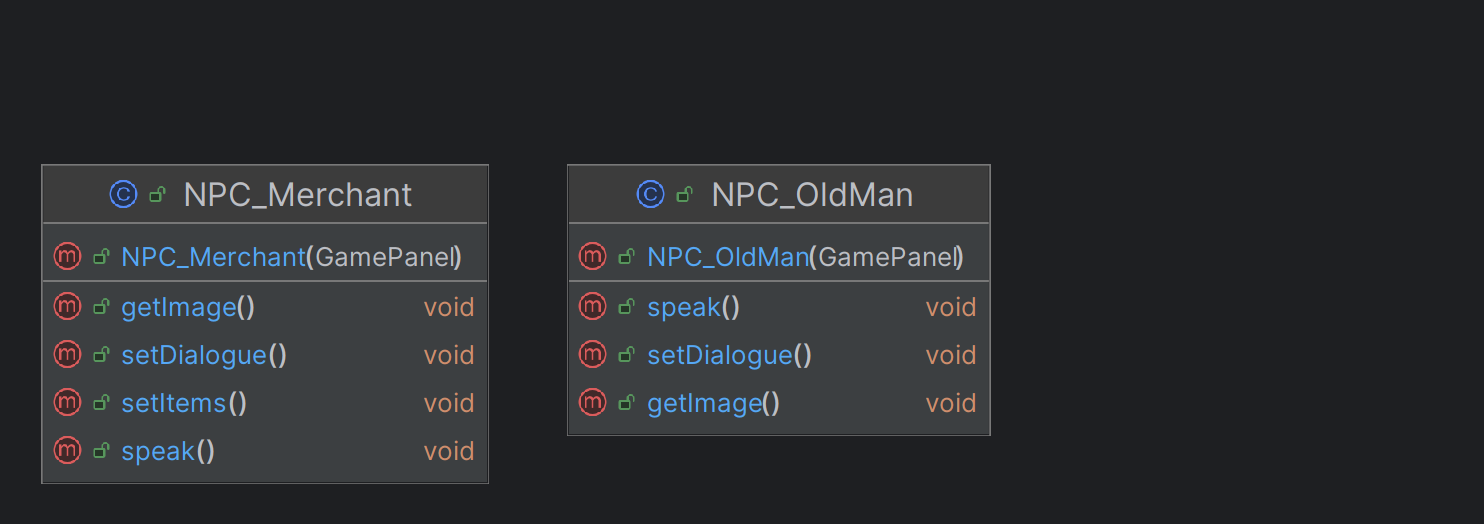


Figure 9: NPC diagram

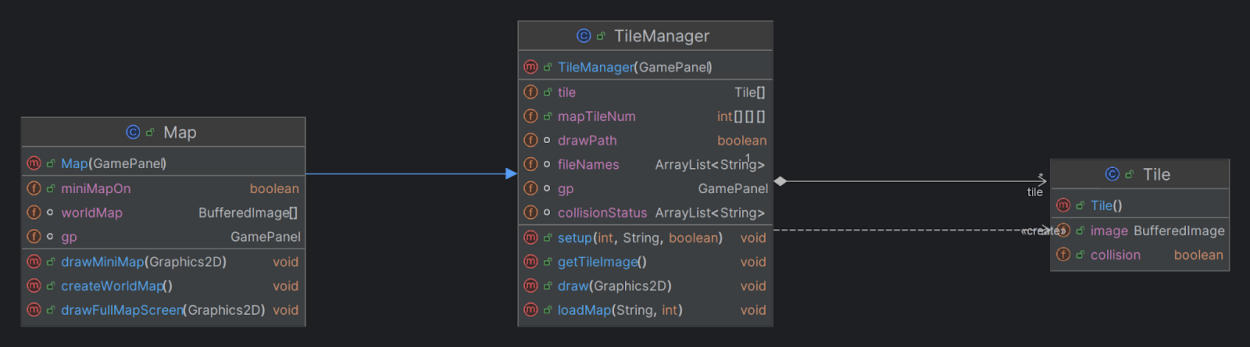


Figure 10: Map diagram

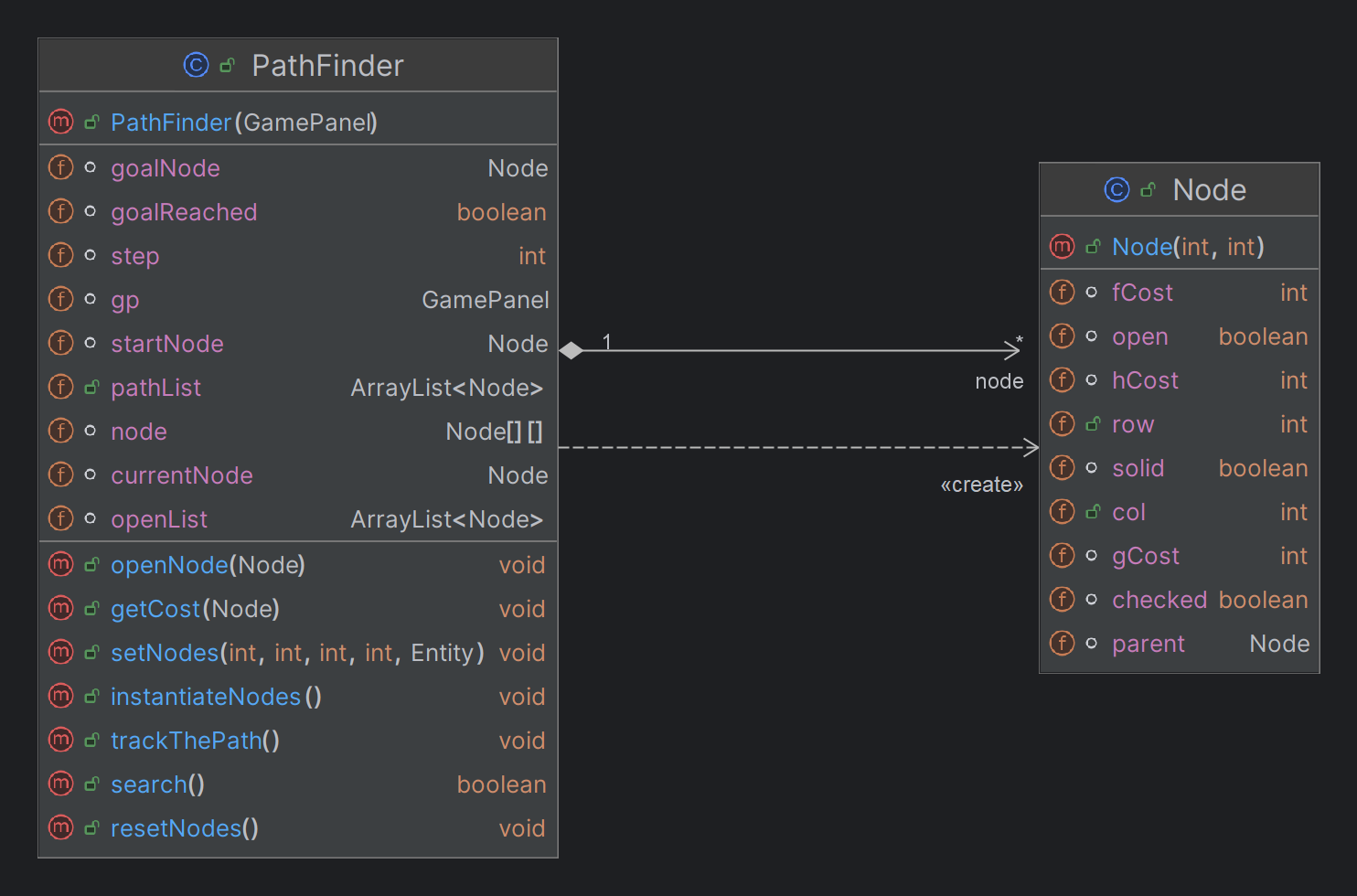


Figure 11: Path finding algorithm diagram

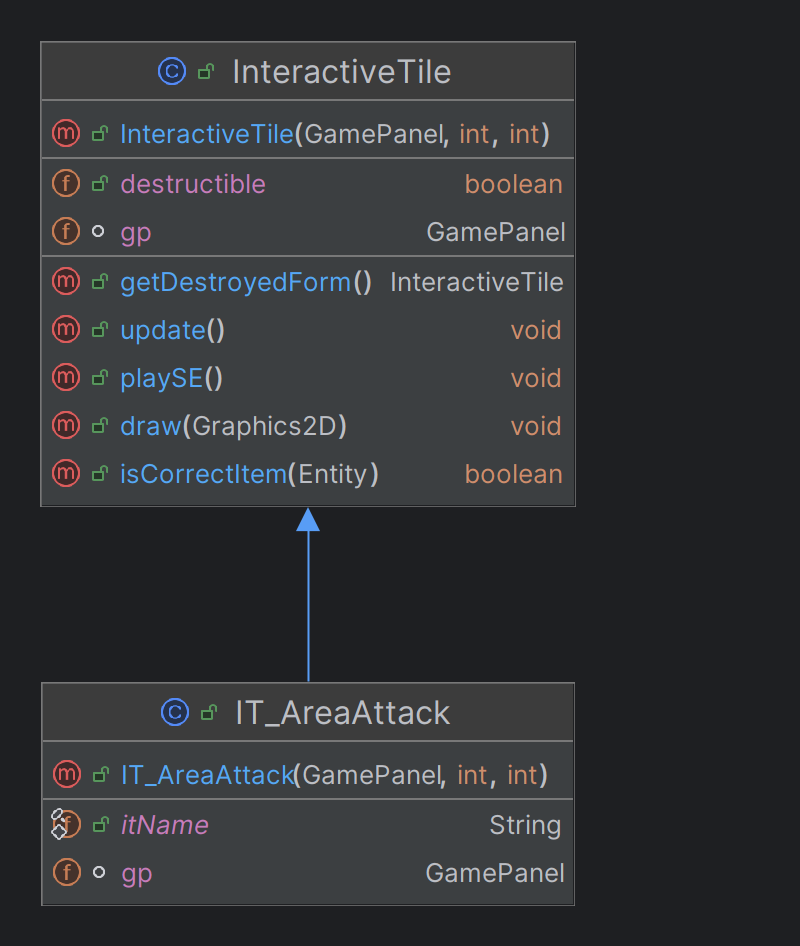


Figure 12. InteractiveTile diagram

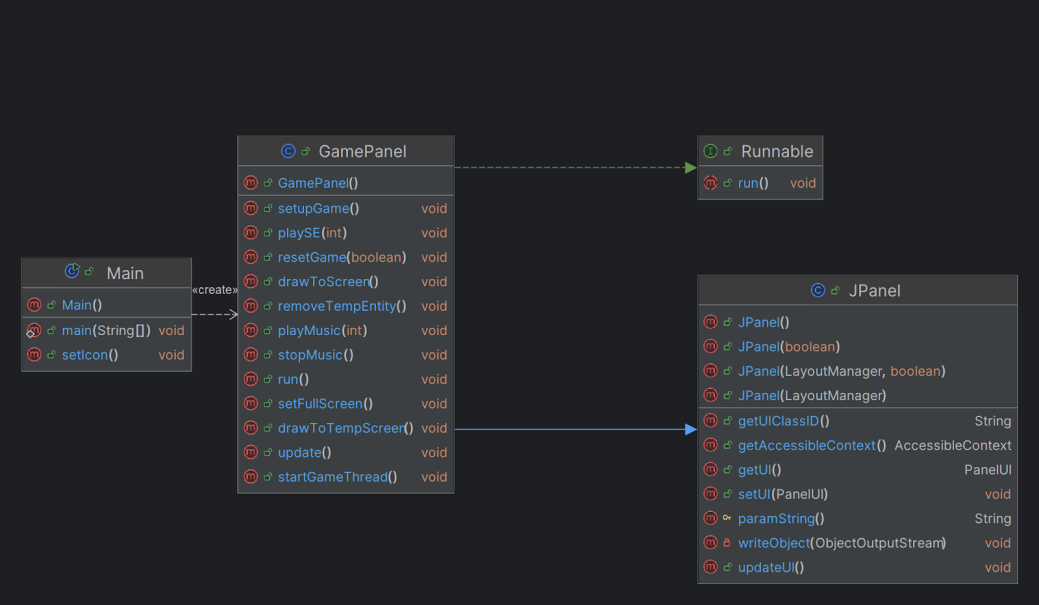


Figure 13: Main diagram

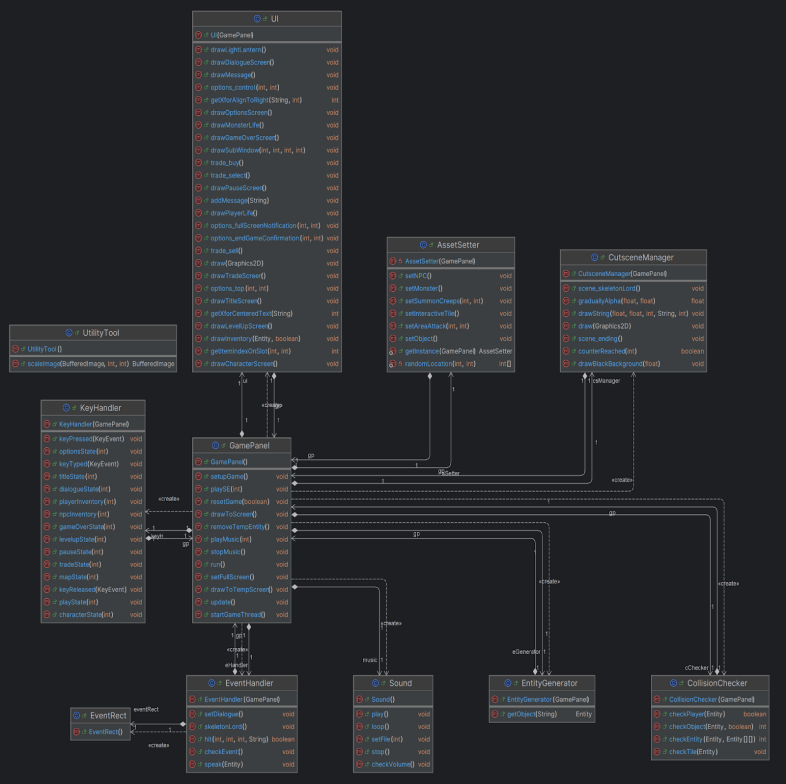
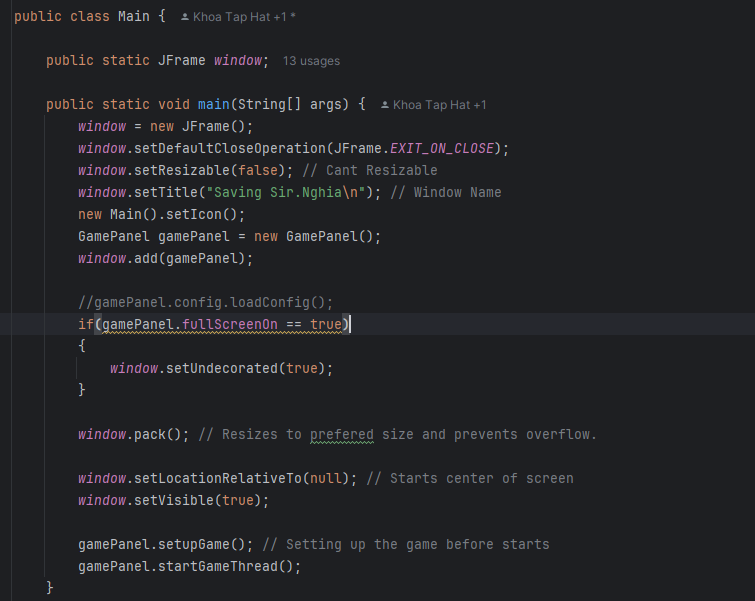


Figure 14: Panel diagram

1. **Code Flow**

Start from class Main, create a screen, and initialize GamePanel. In Main, call the setupGame method of GamePanel to set up the game before it starts. Then call the startGameThread to start the game in GamePanel.

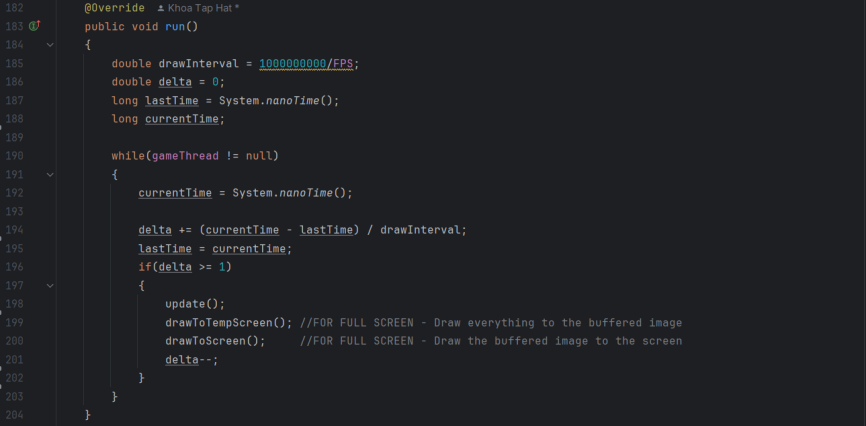


*Figure 15: Main function*



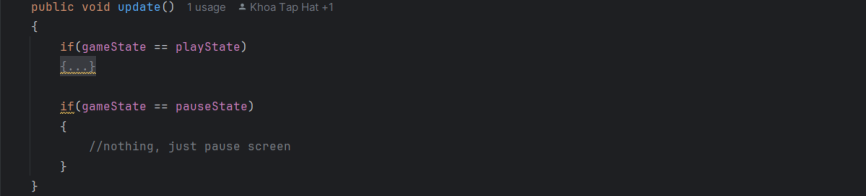
*Figure 16: setupGame function in GamePanel*

In GamePanel, initialize values and lists to run the game in the run method to display on the screen.

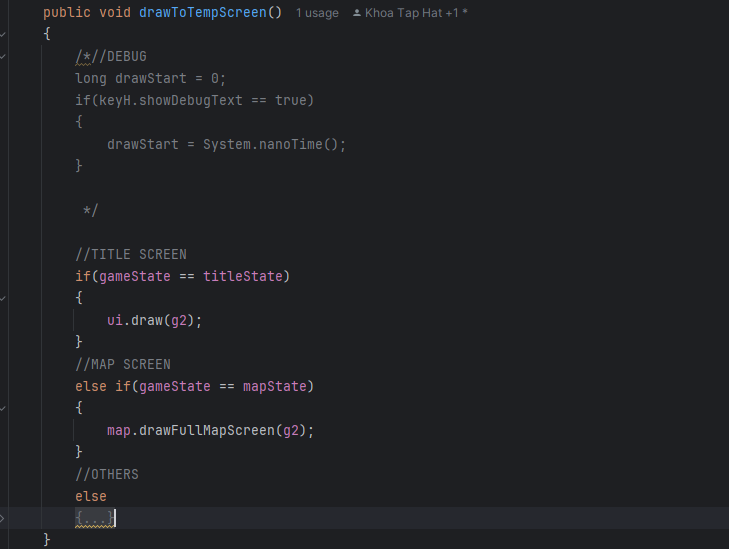


*Figure 17: run function in GamePanel*

In the run method, call update and drawToTempScreen to create a game loop. This loop will update information about the player class, monster, game state, and so on in the update method, and it will continuously draw the game on the screen with drawToTempScreen.



*Figure 18: update function in GamePanel*

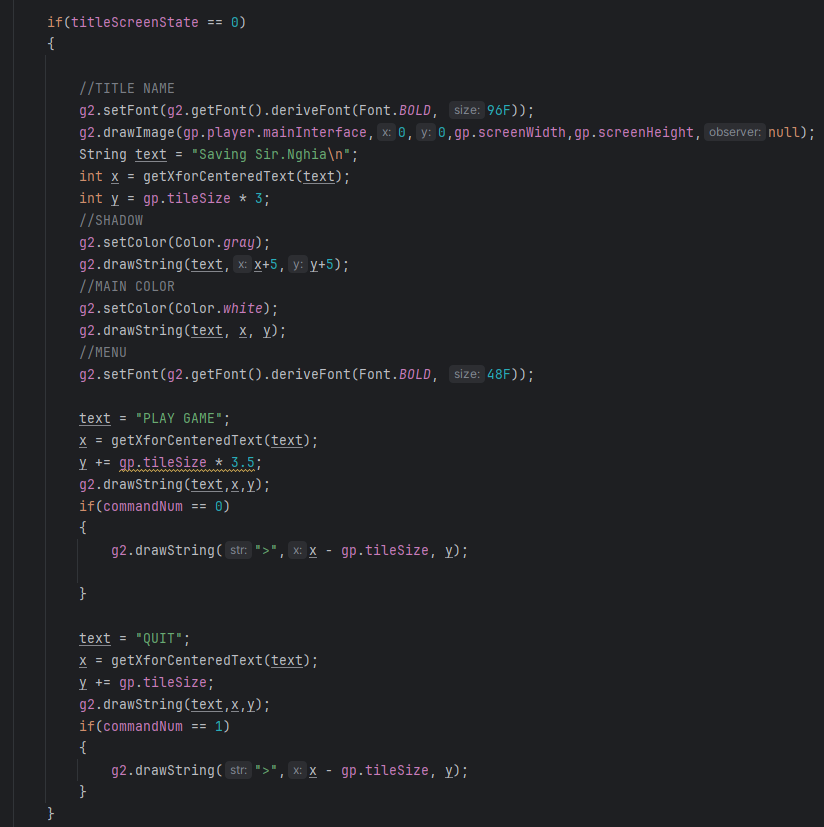


*Figure 19: drawToTempScreen function in GamePanel*



*Figure 20: Game State*

When gameState is titleState, it calls drawTitleScreen to draw the first Title on the screen and keyHandler to handle the button, If you select play, it will switch to the second title screen to select the character.

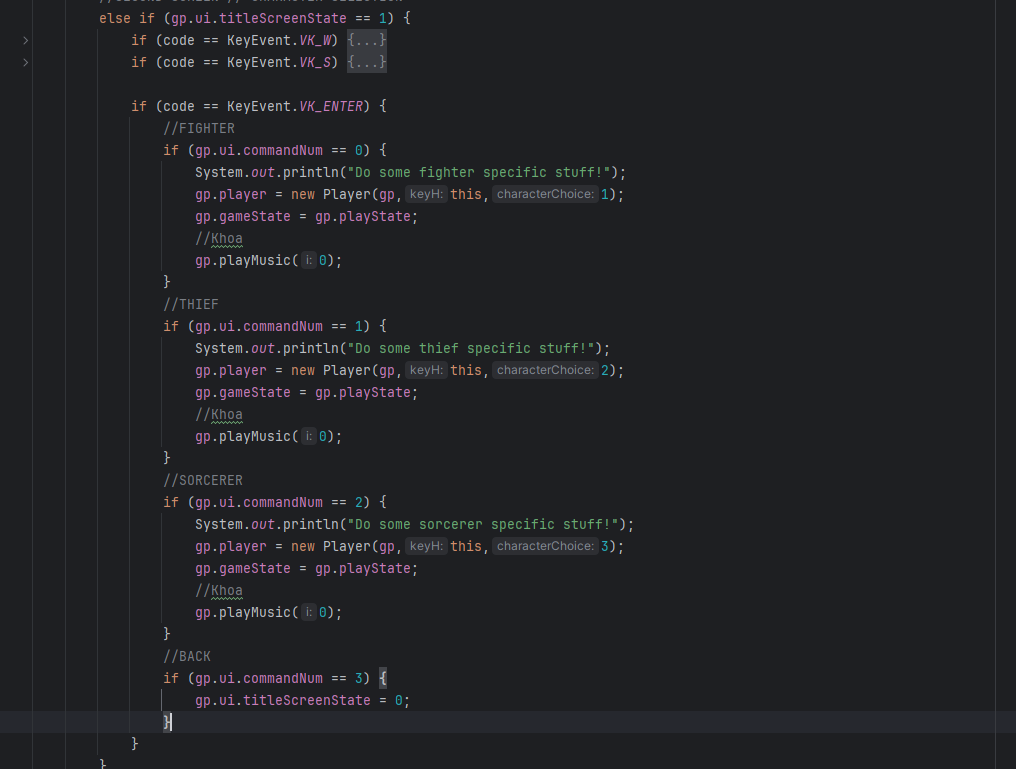


*Figure 21: The 1st title screen in UI class*

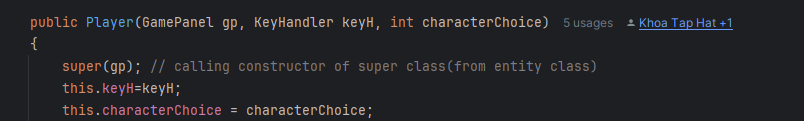


*Figure 22: The 1st title screen in KeyHandler class*

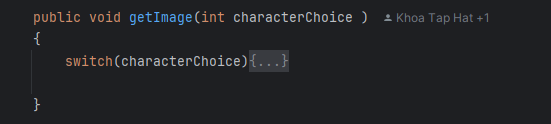
Then the player selects a character, gameState changes to playState, and the KeyHandler retrieves the character's information and transmits it to the Player class. The Player class then uses this information to obtain the appropriate image.



*Figure 23: The 2nd title screen in KeyHandler class*



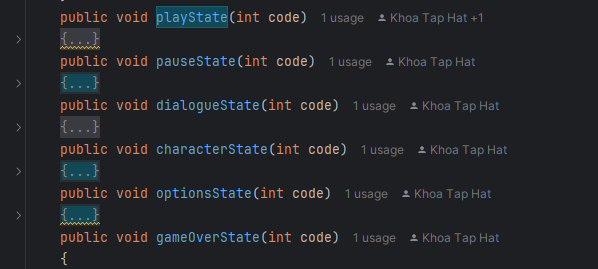
*Figure 24: Constructor of Player class*



*Figure 25: getImage function of Player class*

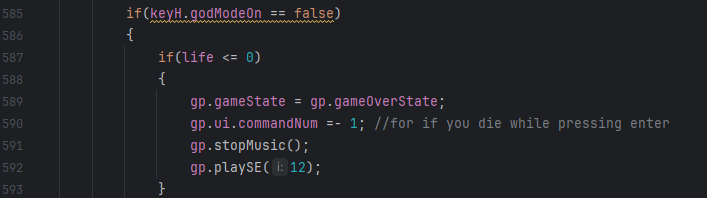
Subsequently, the Player class's update method runs, updating the player's state and animation.

When the game state changes to playState, the keyhandler will handle functions related to the game player such as moving the character, checkLevelup, ....

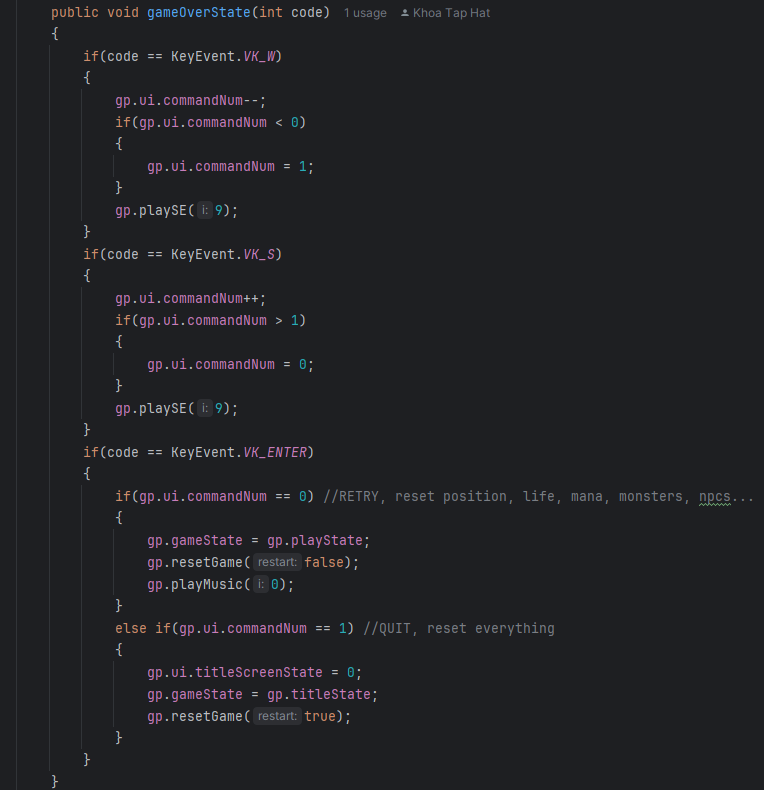


*Figure 26: Functions in playState and so on*

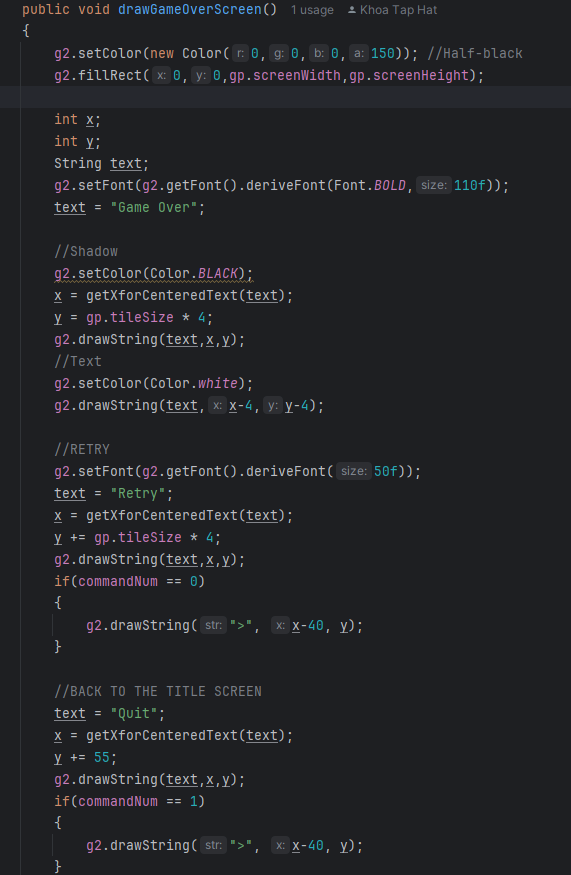
When the player runs out of health, the gameState will switch to gameOverState and call the drawGameOverScreen function to draw the game over window, and the keyhandler to handle button information. If the player chooses to retry, the gameState will return to the playState state, continuing the game, If you choose to quit game, gameState will return the titleState and draw the original screen.



*Figure 27: Condition when the player runs out of health*

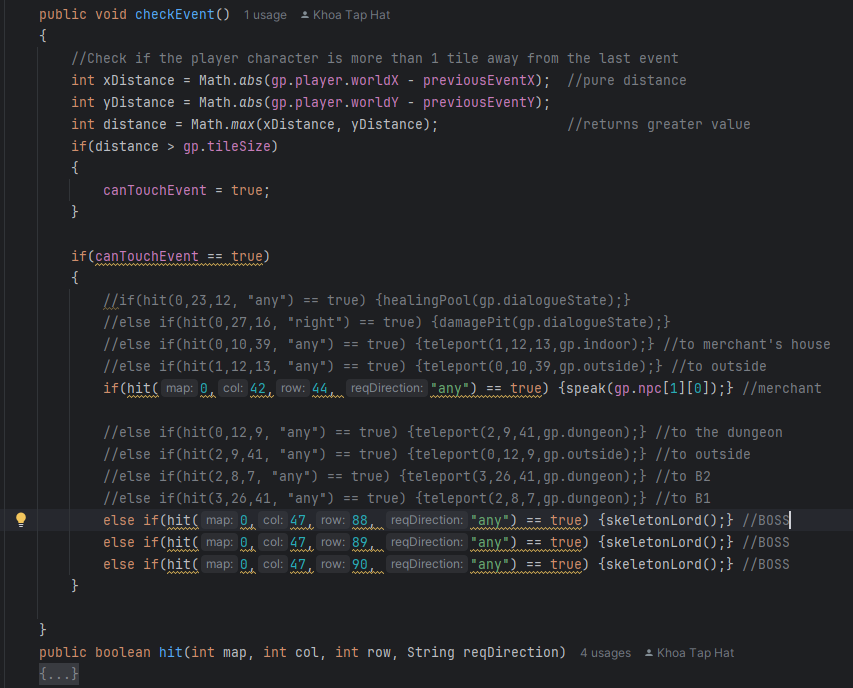


*Figure 28: gameOverState function in KeyHandler class*

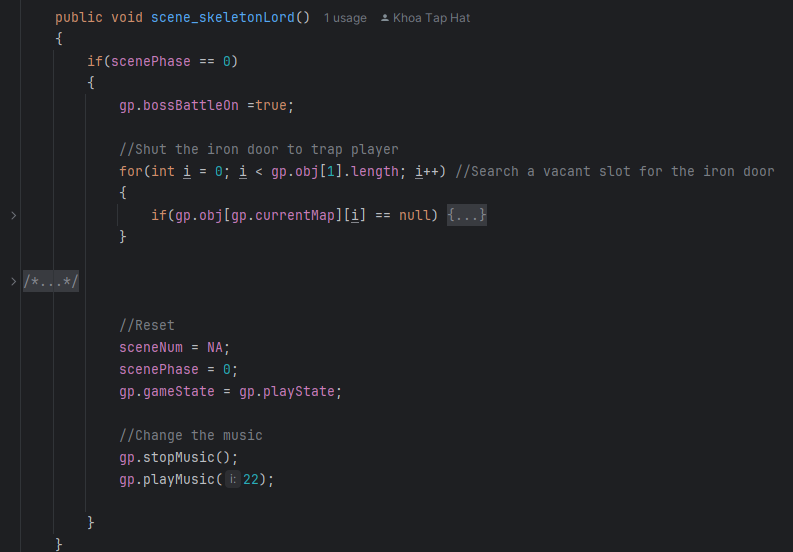


*Figure 29: drawGameOverScreen function in UI class*

When the player hits the checkEvent coordinates in the EventHandler class, the gameState will switch to cutsceneState. In the cutsceneState phase, start with drawing scene\_skeletonLord in CutsceneManager, create objects that overlap the map to prevent the player from leaving and return the gameState state to playState.



*Figure 30: checkEvent function in EventHandler class*



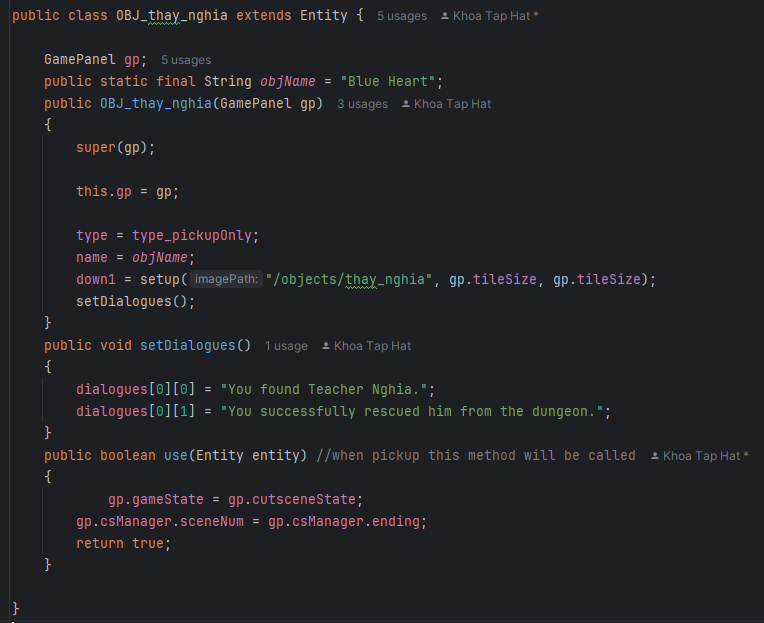
*Figure 31: scene\_skeletonLord function in CutsceneManager class*

When a player kills the boss, the blocking objects will be removed by the checkDrop function of the MON\_SkeletonLord class.

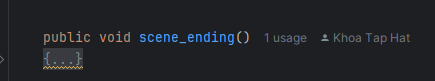


*Figure 32: checkDrop function in MON\_SkeletonLord class*

When the player finds OBJ\_thay\_nghia, the gameState changes to gp.gameState = gp.cutsceneState and calls the scene\_ending function CutsceneManager to to execute the end game scene.



*Figure 33: OBJ\_thay\_nghia class*



*Figure 34: scene\_ending function in CutsceneManager class*

**CHAPTER 3: ENTITY, NPCs, OTHER OBJECT DESIGN**

This is one of the interesting chapters that you may read with enthusiasm because it contains lots of colorful figures. This chapter discusses the characters: the main character, enemies, NPCs, and finally other objects like chests, keys, and doors.

1. **Player**

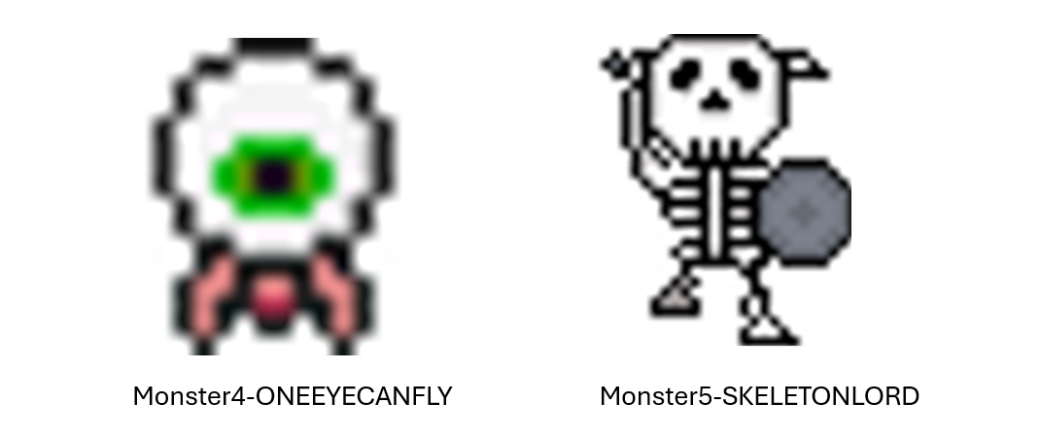


*Figure 35: Main Characters*

The above characters are used for players during the game. With the purpose of diversifying characters and enhancing player experience, we have created 3 characters with 3 completely different formats so players can freely choose their favorite character.

1. **Enemy**





*Figure 36: Enemy Characters*

Monsters in video games serve several purposes. They provide challenges that keep players engaged and excited, offer variety in gameplay that requires strategic thinking, and showcase players' skills and strategic thinking.

For specific,

* BAT: has a high ability in movement that is difficult for player to defeat.
* MINIGROOT: is weakest monster in this game, but the amount of monster is a problem. Player need to handle strong boss while a lot of MINIGROOT is around.
* ORC: is one the the strongest monster with high damage, but restricted movement.
* ONEEYECANFLY: have widely damage that increases challenging and emotion of playing game.
* SKELETON LORD:the final boss with massive damage and dynamic movement. Player need to defeat the final to finish game and save Mr Nghia.

1. **NPCs**



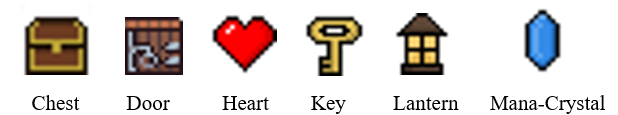
*Figure 37: NPCs*

The NPCs above contain the characters along which pops up with dialogue when player interacts with the NPC. The dialogue system is simple just like other light RPG games.

Specifically:

* The Old Man is a character who provides guidance and direction to the player's character.
* The Merchant is a character who sells items, such as potions or other supplies, to the player.

1. **Other Object**



*Figure 38: Other Object*

The above objects do the following activities when player come in contract:

* Chest: Grants reward to the player whether it will be a coin, weapon,... the first time the player comes in contact.
* Door: Restricts players from entering other areas when they have not found the key.
* Key: To open door.
* Lantern: Helps players see in areas with limited light.
* Heart: Helps players to recovery their missing health.
* Mana-Crystal: Helps players to recovery their missing mana.

**CHAPTER 4: MAP DESIGN**

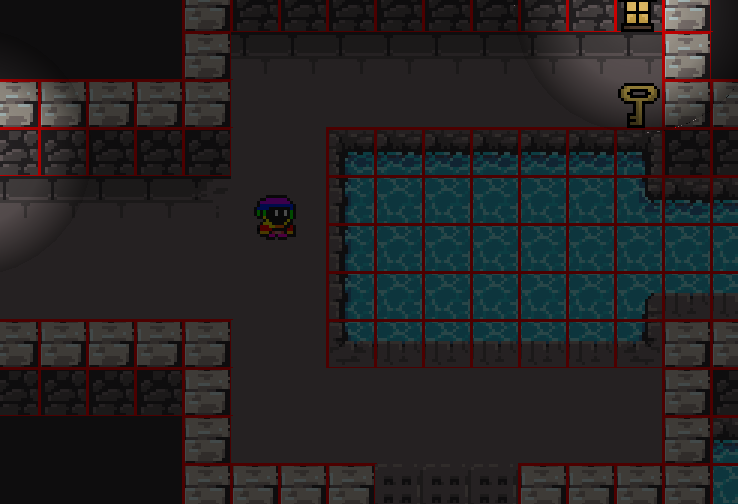
1. **Asset**

We have designed for dungeon for the game using the following asset below:



*Figure 39: Map Tiles[\*]*

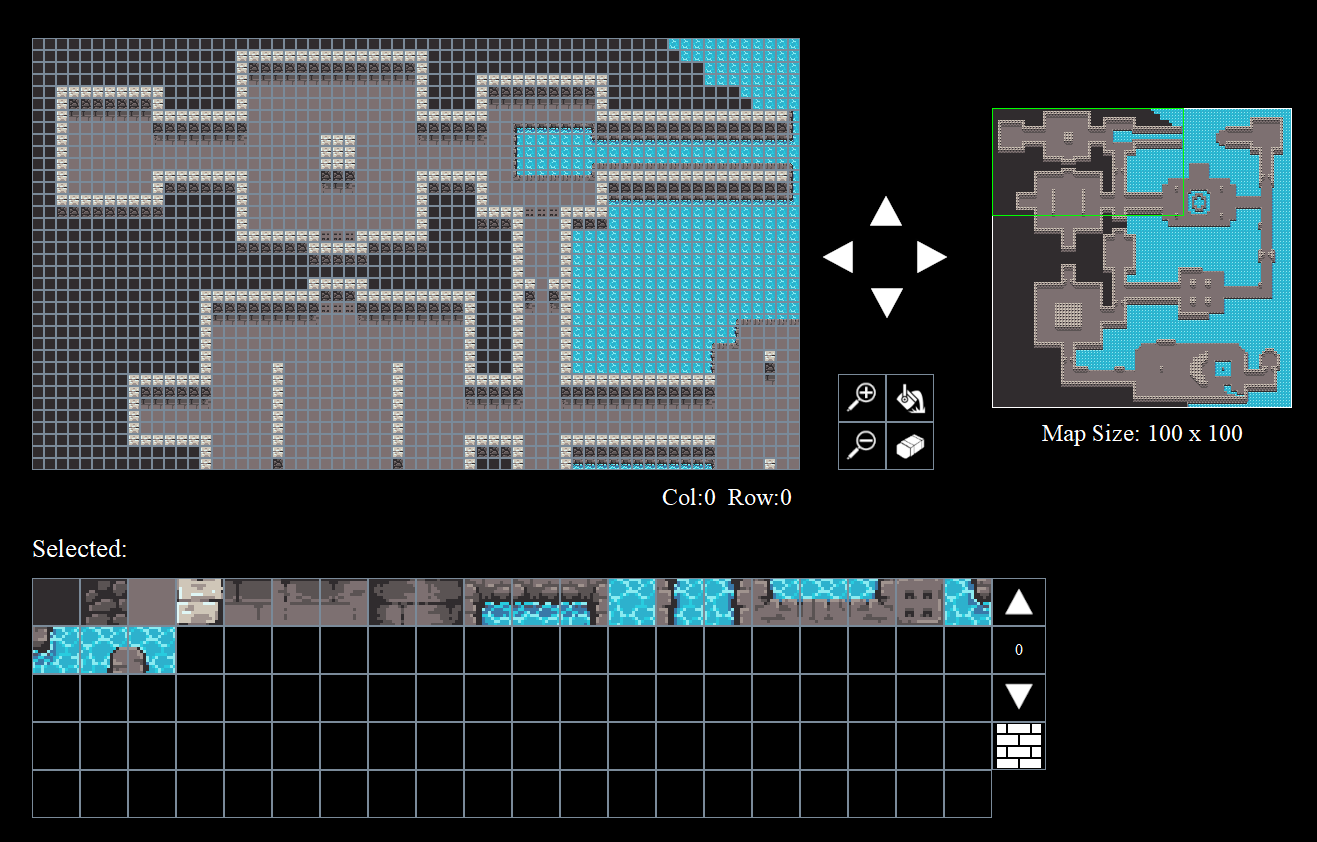
The provided text file "tiledata.txt" contains a list of image filenames paired with boolean values. Each "true" value indicates that the corresponding image represents a 2D box collider in our map design. The red boxes you see below are the 2D box colliders, which ensure that players and enemies stay within the game level.



*Figure 40: 2D Box Colliders*

1. **How To Design**

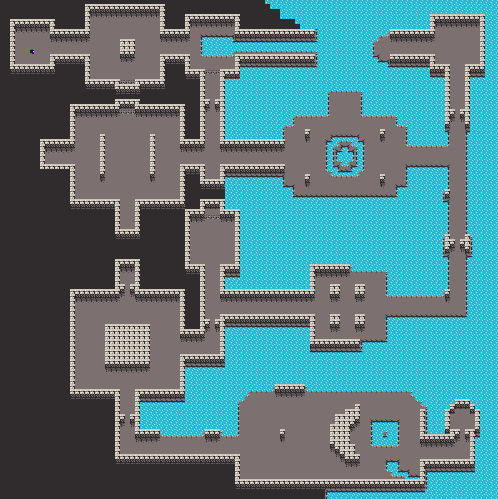
We designed the map using Simple2DTileEditor 1.01 (EXE), a tool that lets you place and draw squares freely. It uses a technique of converting images into digital characters for storage and loading maps.



*Figure 41: Overview Simple2DTileEditor 1.01*

1. **How To Load**

This is described in the System Design Chapter.



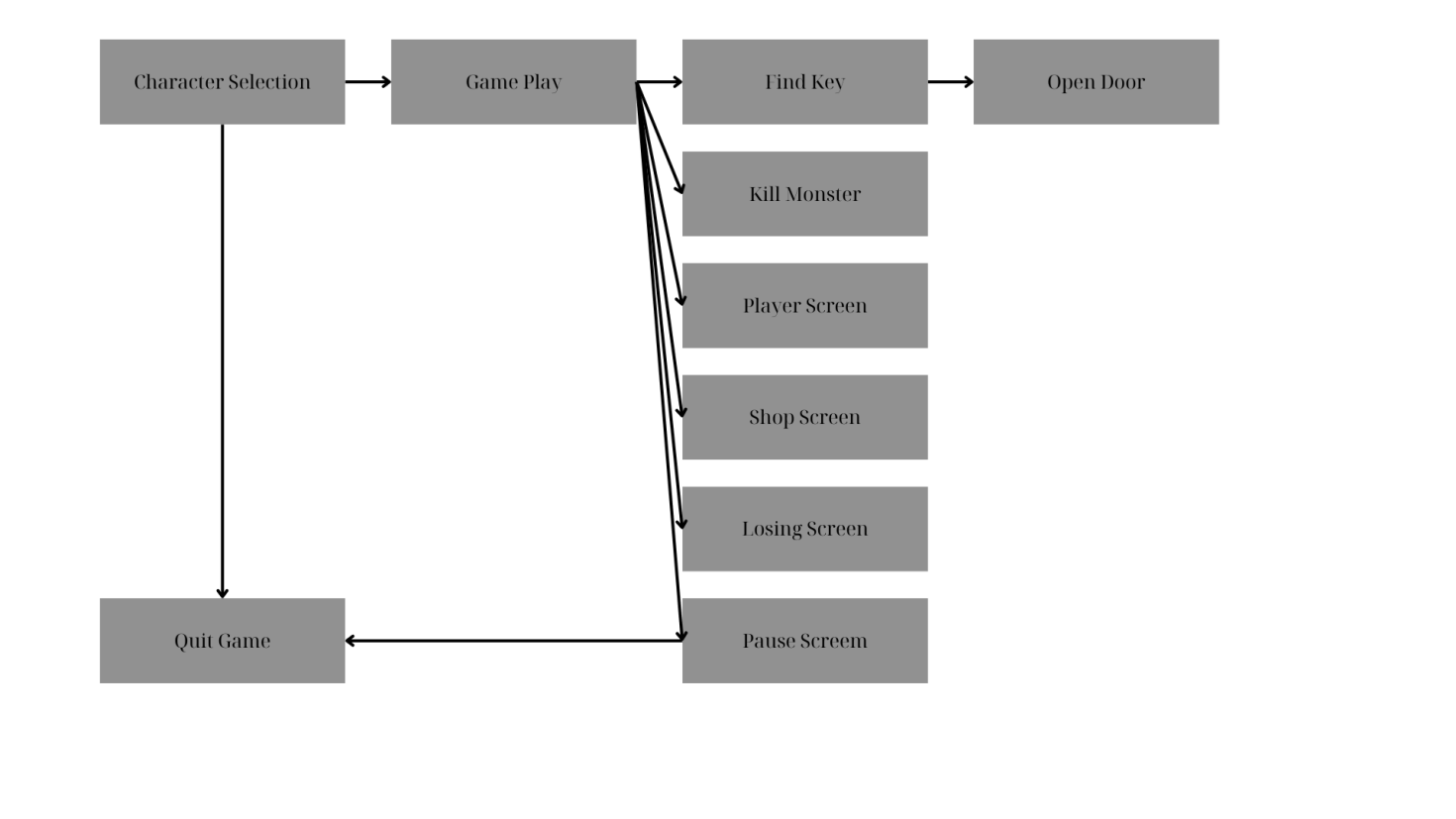
*Figure 42: Overview Map 100x100*

**CHAPTER 5: GAME DESIGN**

1. **Game Rule**

This game is a top-down dungeon game set within a single dungeon. Players control their character to find the key that unlocks the exit door, fight monsters like Bats, Orcs, and..., and ultimately defeat the powerful Boss Skeleton to achieve victory. Each area will contain different types of monsters, each with unique damage and health stats. Defeating monsters rewards players with money, experience, mana, and health restoration. Players can freely explore the dungeon, equipping weapons to enhance their combat abilities. They can also search for and use support items like health and mana restoration. Victory is achieved by defeating the Boss Skeleton and finding Sir. Nghia. However, if the player runs out of health, they lose the game.

1. **Game Structure**



*Figure 43: Game Structure*

1. **Game Play**

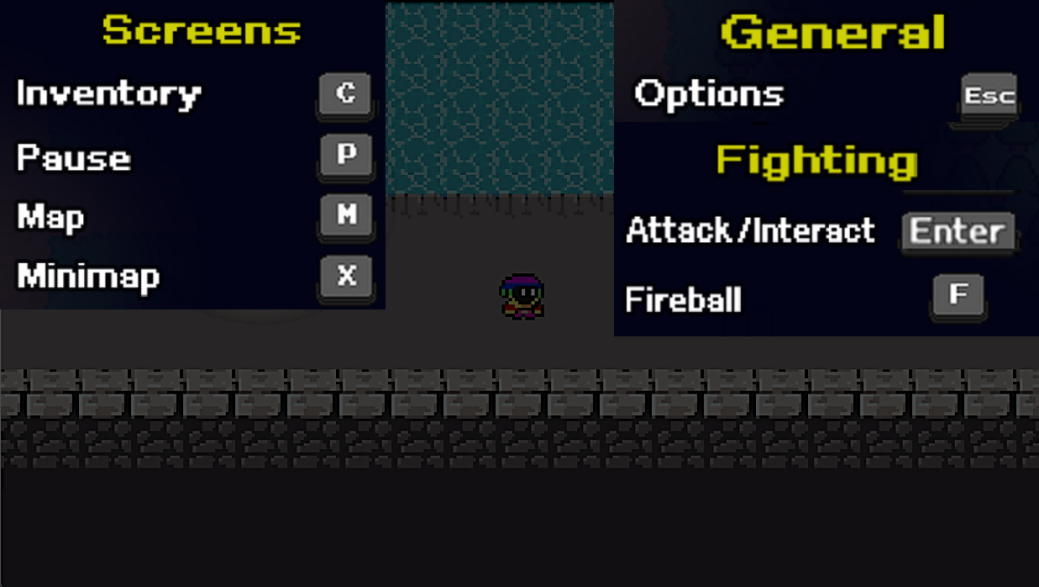
Player Controls:

The following figure illustrates the input to control the player. We use keyboards key such as A,W,S,D to move the player and Enter to attack the enemies.



*Figure 44: Player Controls*

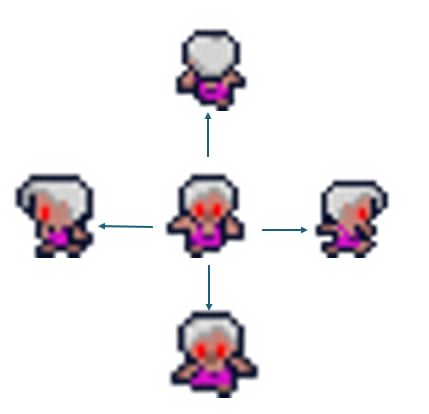
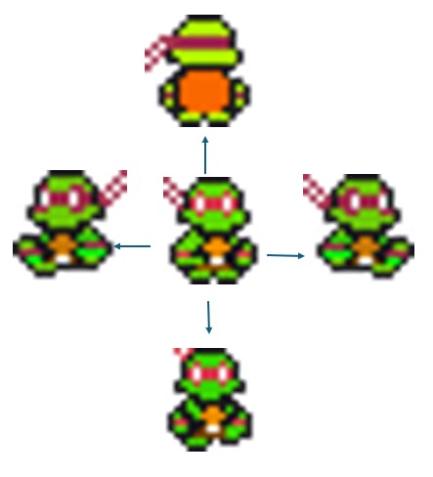
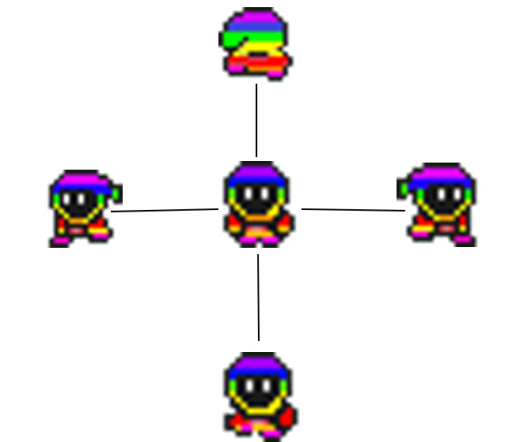
Game Controls:

****

*Figure 45: Game Controls*

1. **Animation States**
   1. Player:

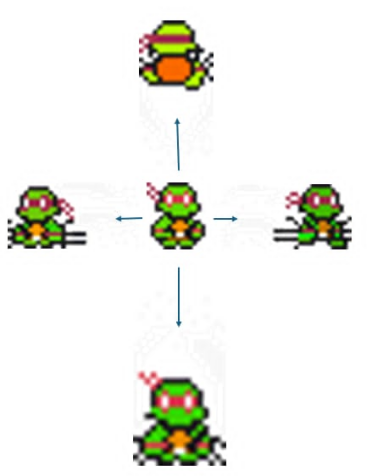
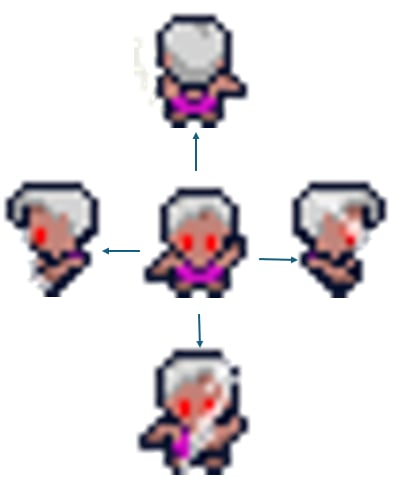
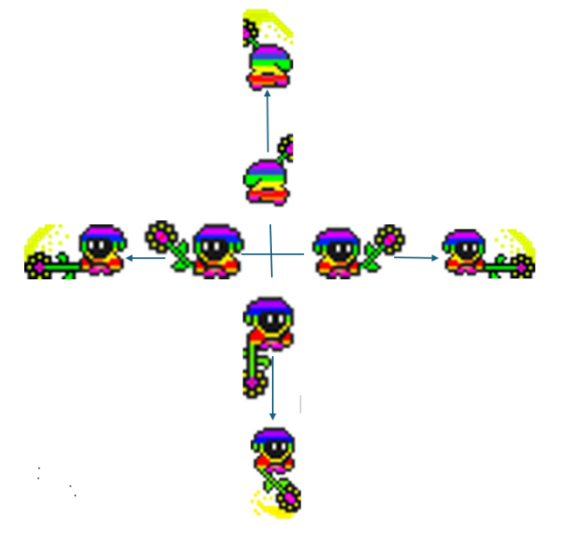
The player has the total of 4 animation states: player\_up, player\_down, player\_left, player\_right, this is a 4-direction movement. We perform player movements by reading frames during each player move to create more realistic movement effects to help players experience the game as realistically as possible. Character orientation also will change based on the movement direction the player chooses. The following image illustrates the idea of directional movements.



*Figure 46: Player Animations*

* 1. Player Attack:

For the player's attack, we also use frames reading for each attack, more specifically 2 images for 4 different directions (left, right, up and down) for every time the player attacks. The images below illustrates two of the player’s four attacking direction, which is attacking left and right sides



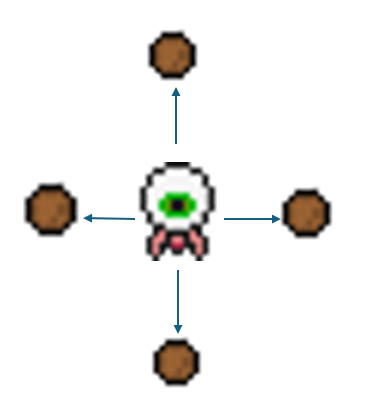
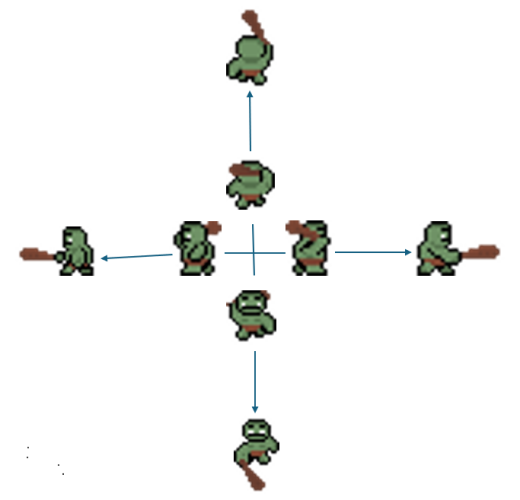
*Figure 47: Player Attack*

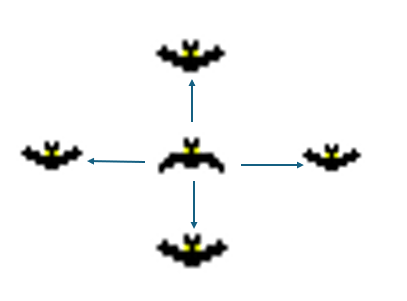
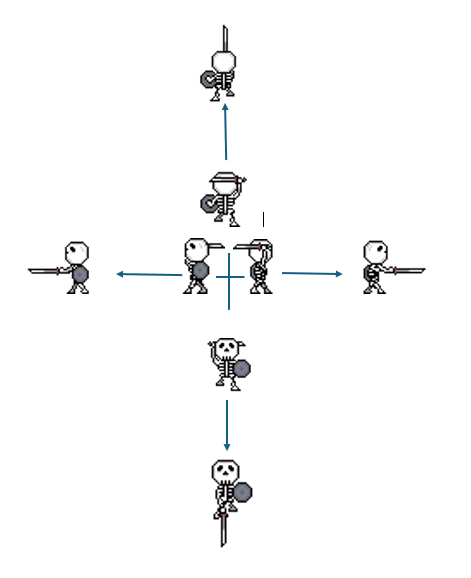
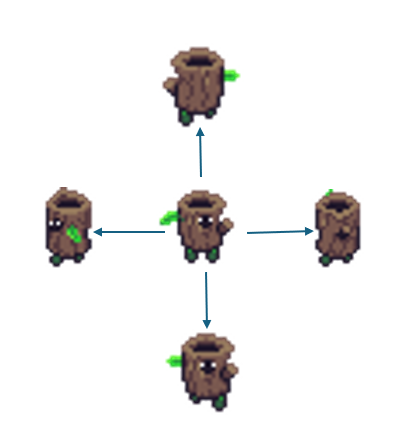
* 1. Monster:

The movement animation of NPCs and monsters is also built similarly to the player, by reading motion frames for moving directions up, down, left, and right.

* 1. Monster Attack:

For the monsters’ attack, we use frames reading for each attack, more specifically 2 images for 4 different directions (left, right, up and down) for every time the player attacks.



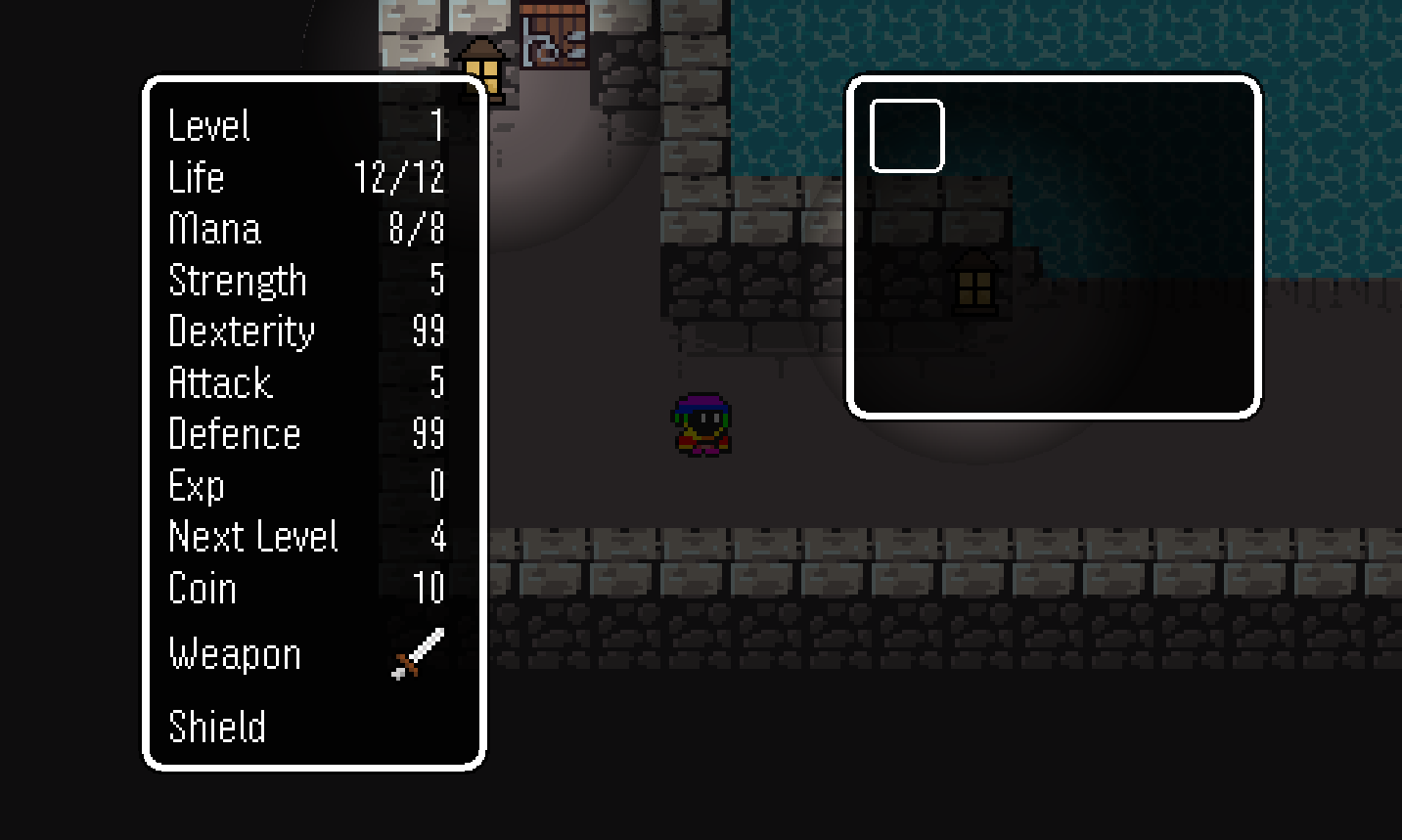


*Figure 48: Monster Attack*

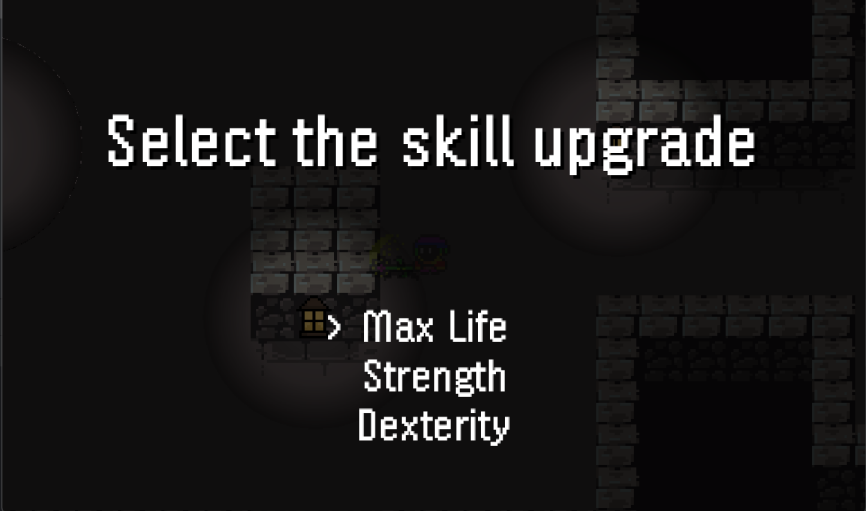
1. **UI**



*Figure 49: Select player*



*Figure 50: Inventory*



*Figure 51: Level up*



*Figure 52: Game pause*



*Figure 53: Game options*



*Figure 54: Game over*

1. **Audio**

The following table shows the list of audio clips with descriptions our have used to develop this game:

|  |  |
| --- | --- |
| Name | Function |
| levelup.wav | Notifies players every time the character levels up |
| hitmonster.wav | Uses every time the player attacks and damages monsters |
| blocked.wav | Notifies the player every time they move and hit an obstacle |
| BlueBoyAdventure.wav | Used throughout the game playing process, as the game's background music, preventing players from getting bored, increasing the drama of the game. |
| burning.wav | Used every time the player is burned by a monster's burning skill |
| coin.wav | Used every time player collecting coin |
| dooropen.wav | Used every time doors are opened |
| fanfare.wav | Used when the player completes all missions and wins the game |
| FinalBattle.wav | Used during the final fight |
| gameover.wav | Used when the player's HP reaches 0 and the character dies |
| powerup.wav | Used after the player has chosen to upgrade the character's power (mana, health, attack power) |
| speak.wav | Used every time the player interacts with NPCs |
| swingweapon.wav | Used as a sword swinging sound every time the player attacks |

*Table 2: List of audio*

**CHAPTER 6: DEMO**

[Click here to view:](https://drive.google.com/file/d/1CLsoQxugP8uQHXMjLca8d3_UI-HAwY3w/view?usp=sharing" \o "Click here to view) https://drive.google.com/file/d/1CLsoQxugP8uQHXMjLca8d3\_UI-HAwY3w/view?usp=sharing

**CHAPTER 7: CONCLUSION AND FUTURE WORKS**

1. **Conclusion**

The development of the game is still ongoing. In the final phase, the team has gained a deeper understanding of the four core principles of Object-Oriented Programming (OOP) and the SOLID principles. This knowledge has significantly enhanced our proficiency in OOP for both game development and post-release programming, introducing novel features compared to the original version. The project classes have covered encapsulation extensively. Inheritance, abstraction, and polymorphism have been most frequently applied within the enemies’ and player packages. Consequently, Saving Mrs Nghia was meticulously developed using the fundamental concepts of OOP, and the game code embodies all four key OOP features and a design pattern learned from class. The extensive knowledge gained from this experience is a testament not only to our collective expertise but also to the innovative spirit that has driven us to push the boundaries of game development.

1. **Future works**

For the timeline and restricted skills , we cannot do all plans that we have in mind. These features could enhance the player’s abilities, special attack, and the new experiences for players including teleport, throwing firearms, and allowing players to customize their character with various skins, outfits, anh jewelries. In technical improvements, performance optimization will be updated to ensure smooth gameplay on various devices. This includes reducing load times, optimization asset usage, and improving frame rates. In content expansion, storyline enhancements will be committed, this would expand the game’s narrative with additional storyline, quests, or side missions.

1. **Acknowledgment**

We would like to convey our deepest appreciation to our instructor and individuals who assisted us in reaching the goals of this project:

* Dr. Tran Thanh Tung and MSc. Nguyen Trung Nghia
* Original code from RyiSnow (https://youtu.be/om59cwR7psI?si=xngu7Kg2FalNV\_4J)

**REFERENCES**

1. *BlueBoyAdvanture (2022).RyiSnow/blueboyadvanture[Java]. Retrieved May 10, 2024, from*

*<https://github.com/berkayw/Blue-Boy-Adventure.git>*

*[2] Java Swing Tutorial. Javatpoint. (n.d.). Retrieved December 11, 2023, from*

https://www.javatpoint.com/java-swing

*[3] UML Diagrams Tutorial. Javatpoint. (n.d.). Retrieved January 4, 2023, from*

https://www.javatpoint.com/uml-diagrams

*[4] Drew, O. (2022). Othneildrew/Best-README-Template. Retrieved January 4,*

*2023, from*

https://github.com/othneildrew/Best-README-Template

*[5] SOLID Principle in Programming: Understand With Real Life Examples.*

*GeeksforGeeks. Retrieved December 28, 2023, from*

<https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/>

*[\*] aztharis.itch.io/tilemap-mini-dungeon, from*

https://aztharis.itch.io/tilemap-mini-dungeon