VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

PROJECT REPORT

FINANTASY

OBJECT-ORIENTED PROGRAMMING (IT069IU)

Course by

Dr. Tran Thanh Tung and MSc. Nguyen Trung Nghia

Members_ID Student

Trần Quốc Anh - ITITIU22006

Phan Trần Thanh Huy - ITCSIU22056

Nguyễn Đình Khánh Ngân - ITCSIU22236

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

1. Gaming in the Field

Video game creation is a unique and rapidly expanding industry within software engineering and development. Unlike other software projects, it involves the collaboration of teams from diverse disciplines such as acting, music, art, and programming. The process of creating a compelling game experience relies heavily on prototypes and iterations. For our four-credit "Object-Oriented Programming" course, which counts toward our degree, we have chosen to develop a turn-based game.

This project allows us to improve the development cycle, reduce development time, and enhance graphics. It also provides us with the opportunity to apply the theories we have learned in class and hone our Java skills, particularly in object-oriented programming and design patterns.

2. About the game project

Turn-based games are a subset of strategy games, characterized by the players' autonomous decision-making skills that significantly influence the game's outcome. These games require internal decision tree-style thinking and typically demand a high level of situational awareness.

Based on this concept, we decided to develop a turn-based game with a modern user interface, additional weapons, and monsters with numerous special abilities. Our game, inspired by the classic 2048, aims to be more user-friendly while paying homage to earlier games.

3. Our game project

Our inspiration came from one of the most well-known turn-based games: Square Enix's Final Fantasy I (1987). Our game incorporates all the fundamental elements of a Final Fantasy game, where players take turns performing their actions. Each player has a designated period during which they can make their moves, decisions, or actions. Once a player's turn is completed, the next player takes their turn. This cycle continues until the game's objective is met or the game concludes according to its rules.

We have made several improvements to enhance the gaming experience:

- **Adjusting Character Attributes**: Adjusted the ability and attribute values of characters to provide a balanced and engaging gameplay experience.
- **Enhancing Aesthetics**: Improved the game's aesthetics by including additional sounds and animations, making the game visually appealing and immersive.
- **Applying Programming Techniques**: Applied advanced object-oriented programming skills such as the factory pattern and adapter pattern to ensure a robust and scalable game design.

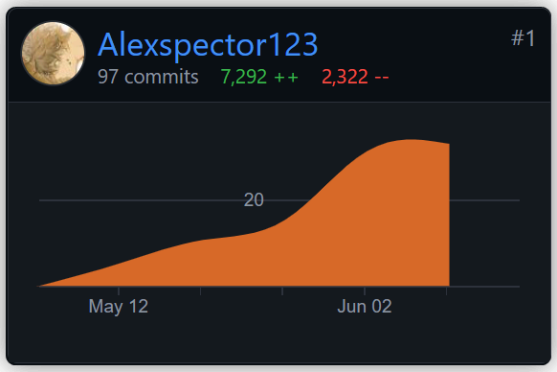
4. References

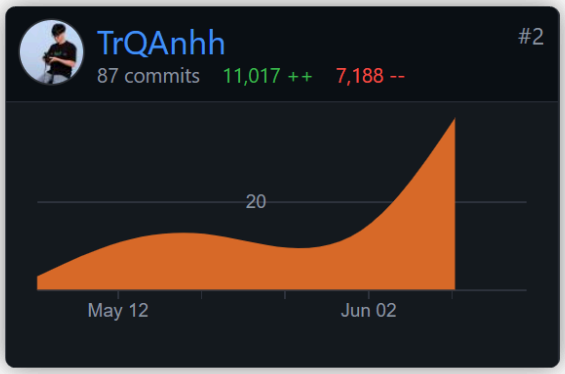
- Images from:
  - https://www.pinterest.com/
  - Top game assets - itch.io
  - https://stackoverflow.com/
- How to make a 2D Game in Java
  - How to Make a 2D Game in Java
  - Object Oriented Programming (OOP) in Java Course
- Turn-based Game idea:
  - Tutorial Series: Turn-Based RPG
  - How Do You Improve Turn-Based Combat?
  - Make a Turn-based RPG Combat Tutorial
  - More Engaging Turn-Based Combat in RPGs
  - Final Fantasy 1 100% walkthroug

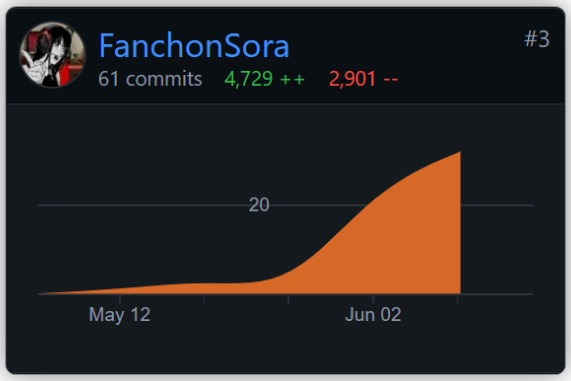
5. Developer team

Our Object-Oriented Programming team comprises three members from International University, including two majoring in Computer Science and one in Information Technology.

Name - Github	Student ID	Jobs and tasks	Contribution
Tran Quoc Anh TrQAnhh	ITITIU22006	Drawing, character and monster graphics, sound setting, map design, design UI, fix bug, design trade shop, design weapon and other inventory	34%
Nguyen Dinh Khanh Ngan FanchonSora	ITCSIU22236	Design pattern monster in cave map and boss monster event, fix bug, design ending and credit, design UI, battle boss system design	32%
Phan Tran Thanh Huy Alexspector123	ITCSIU22056	design battle system, design UI, fix bug, design monster outside cave event, design inventory system, trade shop, effect setting	34%







Chapter 2. SOFTWARE REQUIREMENTS

1. What we have

1. Minimum maintenance cost (graphics).
2. Simple to use.
3. Expected requirements are available in the PC setup.
4. Using professional experience and considered coding.

2. What we desire

1. Develop a game at a reasonable cost.
2. High definition.
3. Simple to update.
4. Create the game in an effective way.

3. Working tools, platforms

1. Visual Studio Code with addition library
    a. JRE System Library [JavaSE-17]
implementation 'Eclipse Temurin JDK with Hotspot 17.0.3+7 (x64)'
2. IntelliJ IDEA for Java and Kotlin
    a. JRE System Library [JavaSE-17]
implementation 'Eclipse Temurin JDK with Hotspot 17.0.3+7 (x64)'
3. Adobe Illustrator 2023
4. Adobe Photoshop 2023

4. Use Case Scenario

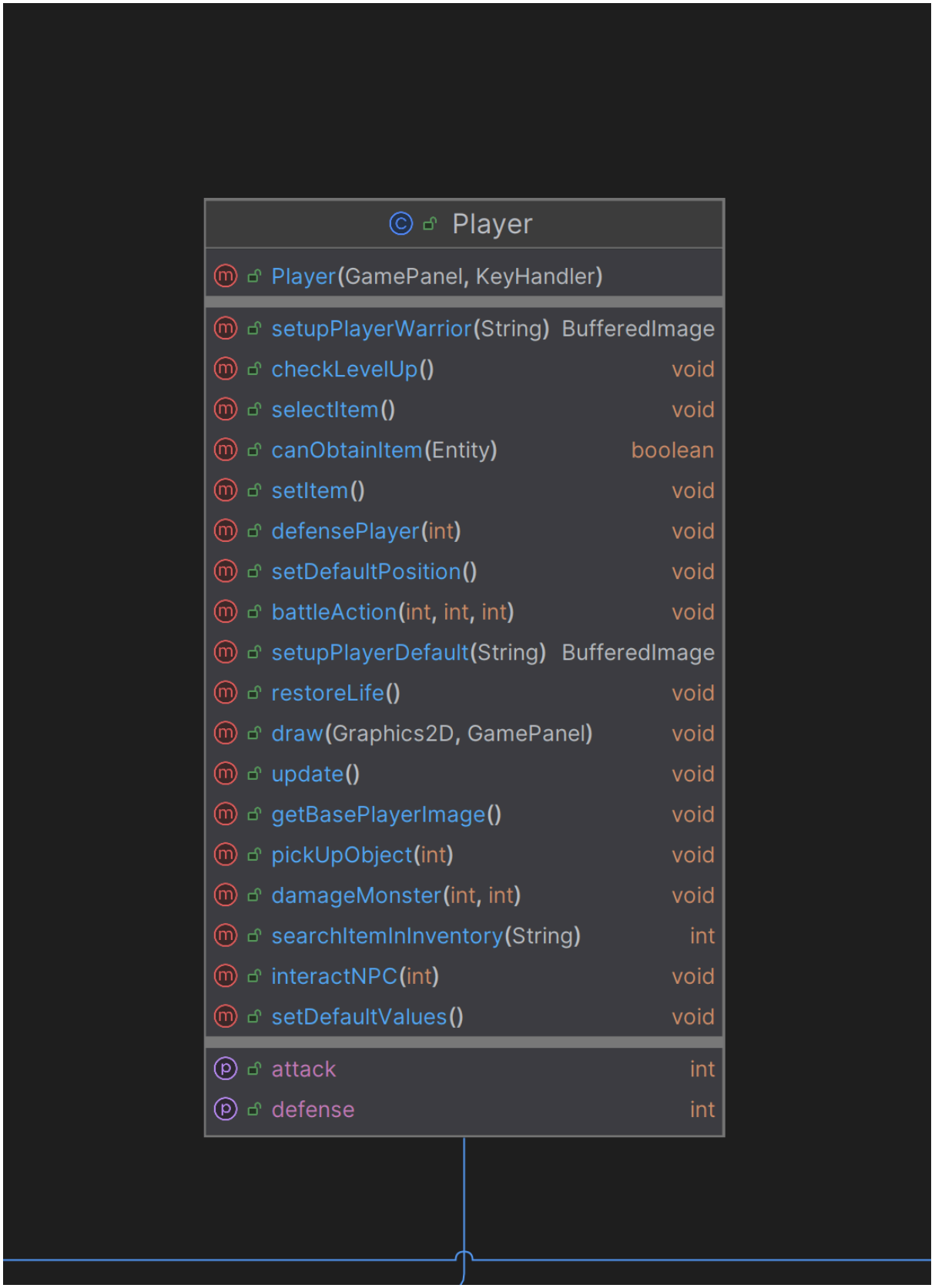
We have developed the use cases according to the game mechanics.

FINANTASY	PLAY	Play the game
		Explore the game
	TASKS	Do game's side tasks
	TRADE	Buy the equipments
	FIGHT	Fight the monsters
	PLOT	Complete the story path

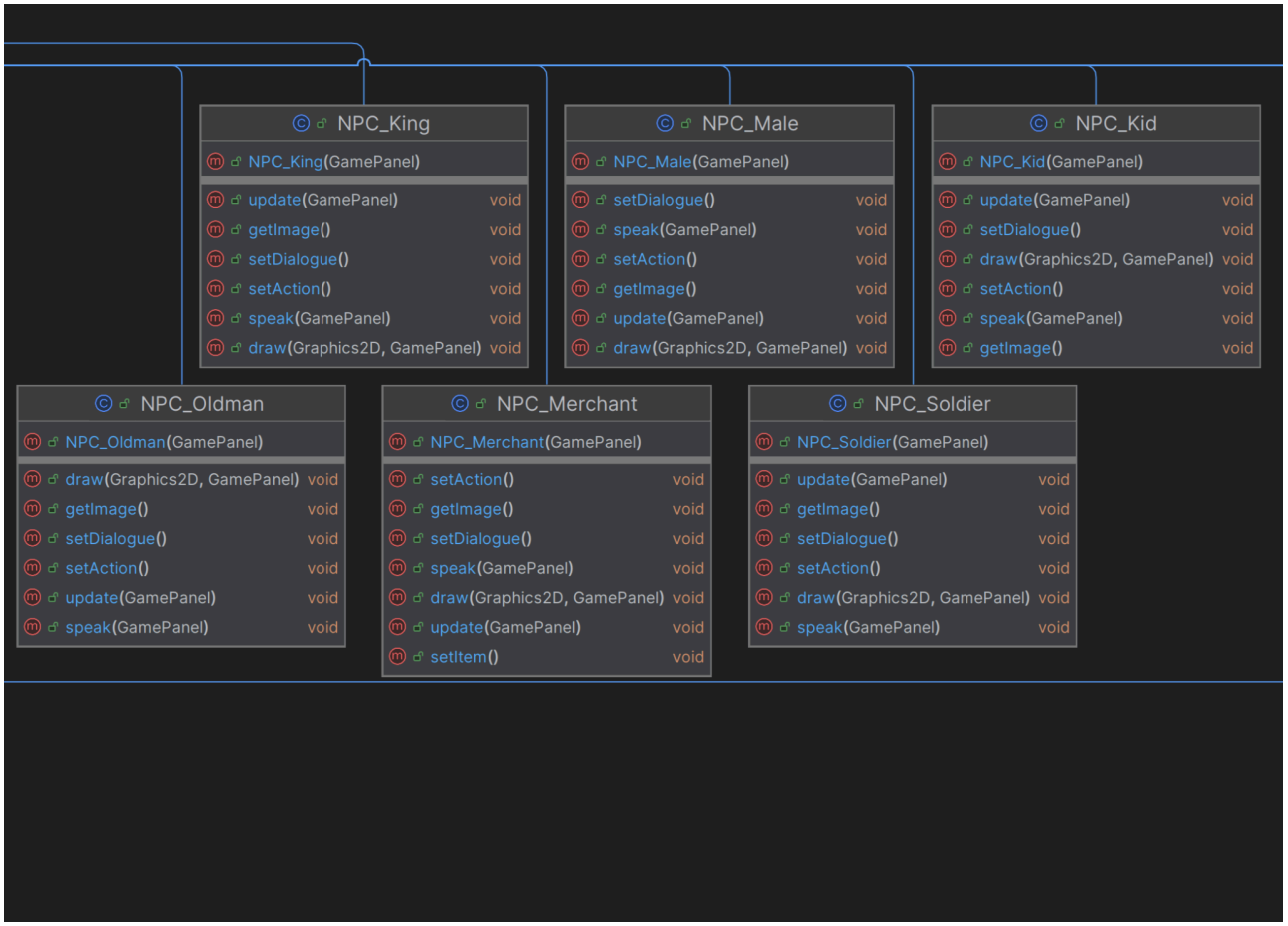
5. Class diagram

Play process diagram

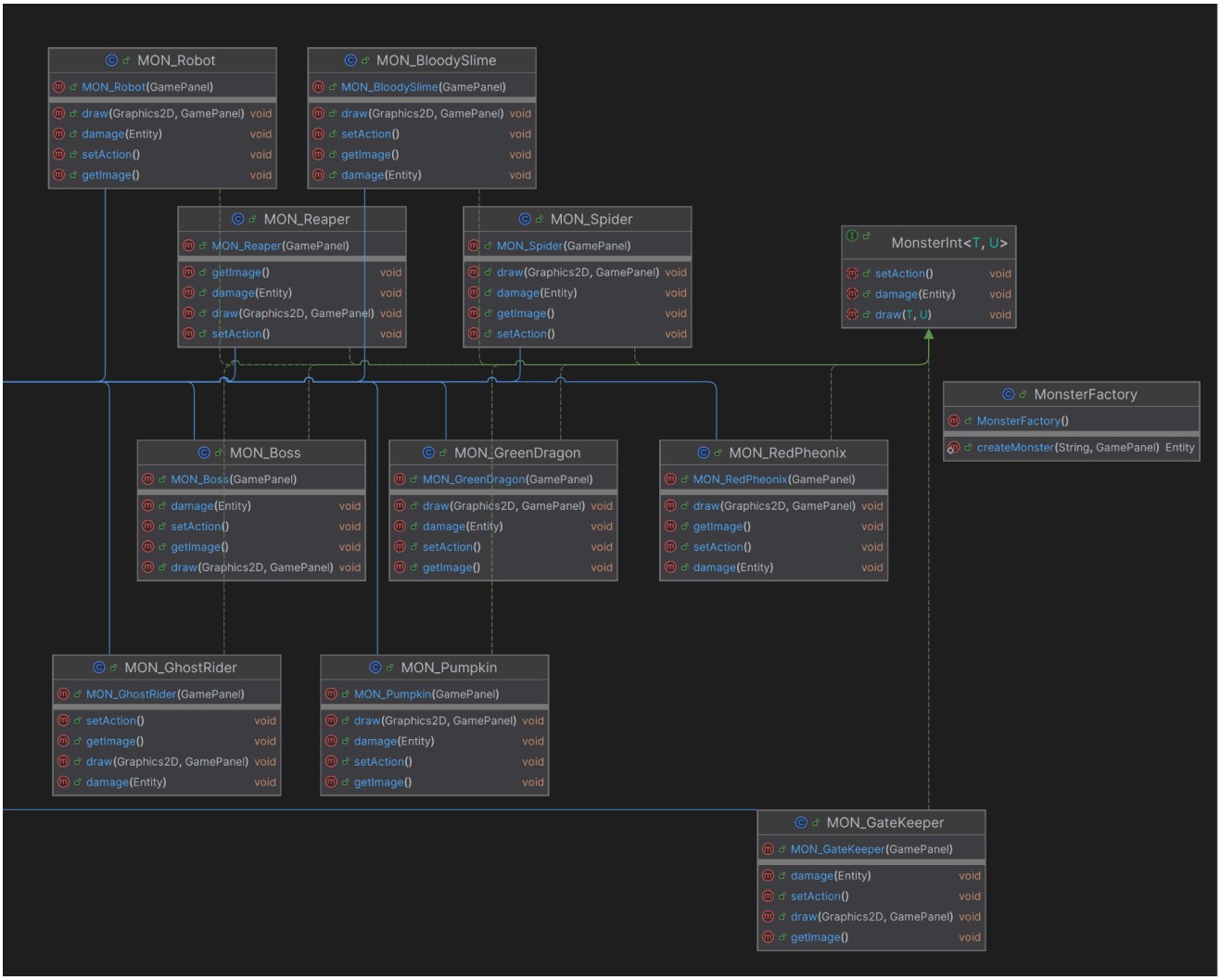
Player diagram



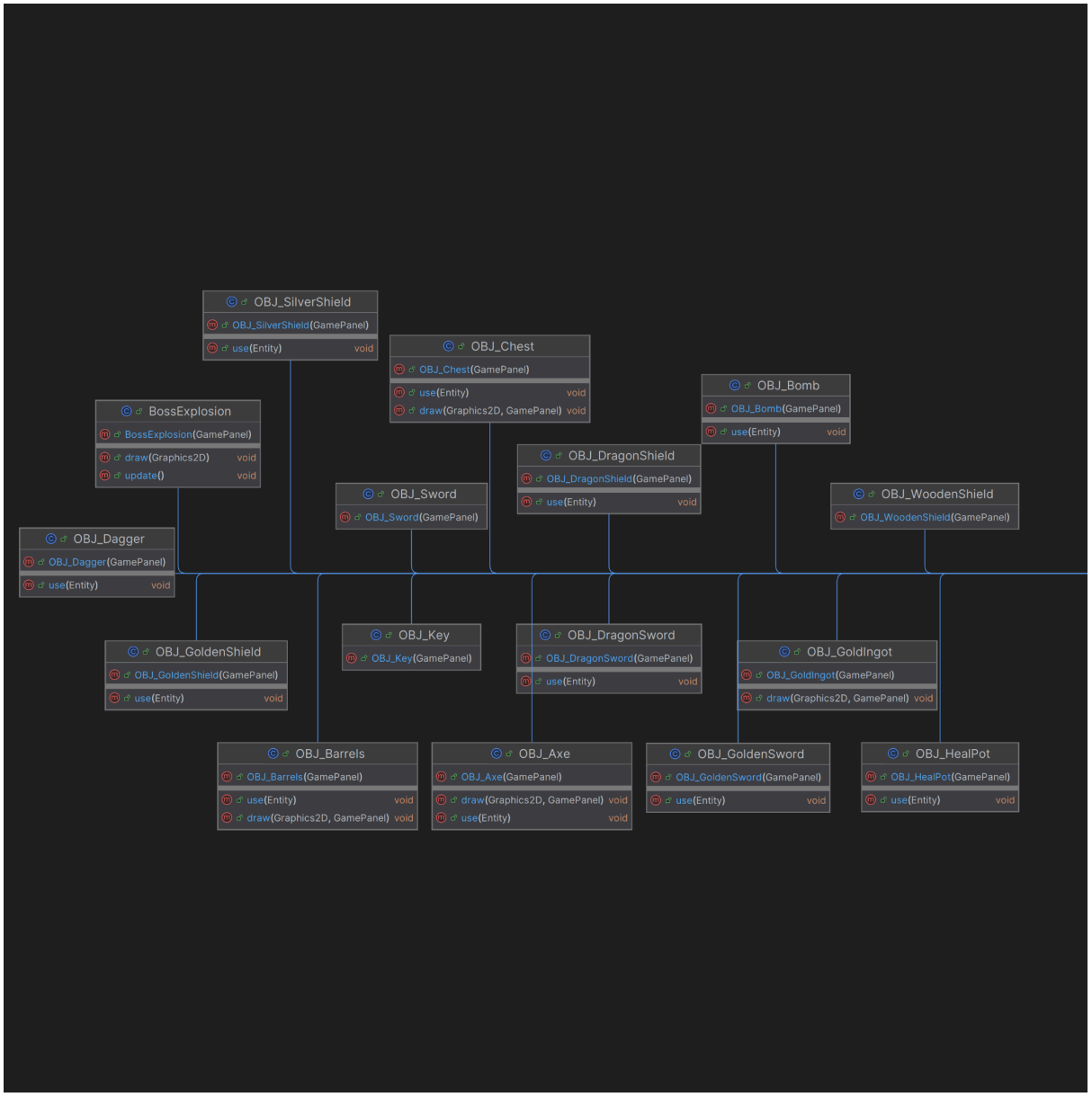
NPC diagram


NPC_King
NPC_King(GamePanel)
update(GamePanel) void
getImage() void
setDialogue() void
setAction() void
speak(GamePanel) void
draw(Graphics2D, GamePanel) void
NPC_Male
NPC_Male(GamePanel)
setDialogue() void
speak(GamePanel) void
setAction() void
getImage() void
update(GamePanel) void
draw(Graphics2D, GamePanel) void
NPC_Kid
NPC_Kid(GamePanel)
update(GamePanel) void
setDialogue() void
draw(Graphics2D, GamePanel) void
setAction() void
speak(GamePanel) void
getImage() void
NPC_Oldman
NPC_Oldman(GamePanel)
draw(Graphics2D, GamePanel) void
getImage() void
setDialogue() void
setAction() void
update(GamePanel) void
speak(GamePanel) void
NPC_Merchant
NPC_Merchant(GamePanel)
setAction() void
getImage() void
setDialogue() void
speak(GamePanel) void
draw(Graphics2D, GamePanel) void
update(GamePanel) void
setItem() void
NPC_Soldier
NPC_Soldier(GamePanel)
update(GamePanel) void
getImage() void
setDialogue() void
setAction() void
draw(Graphics2D, GamePanel) void
speak(GamePanel) void

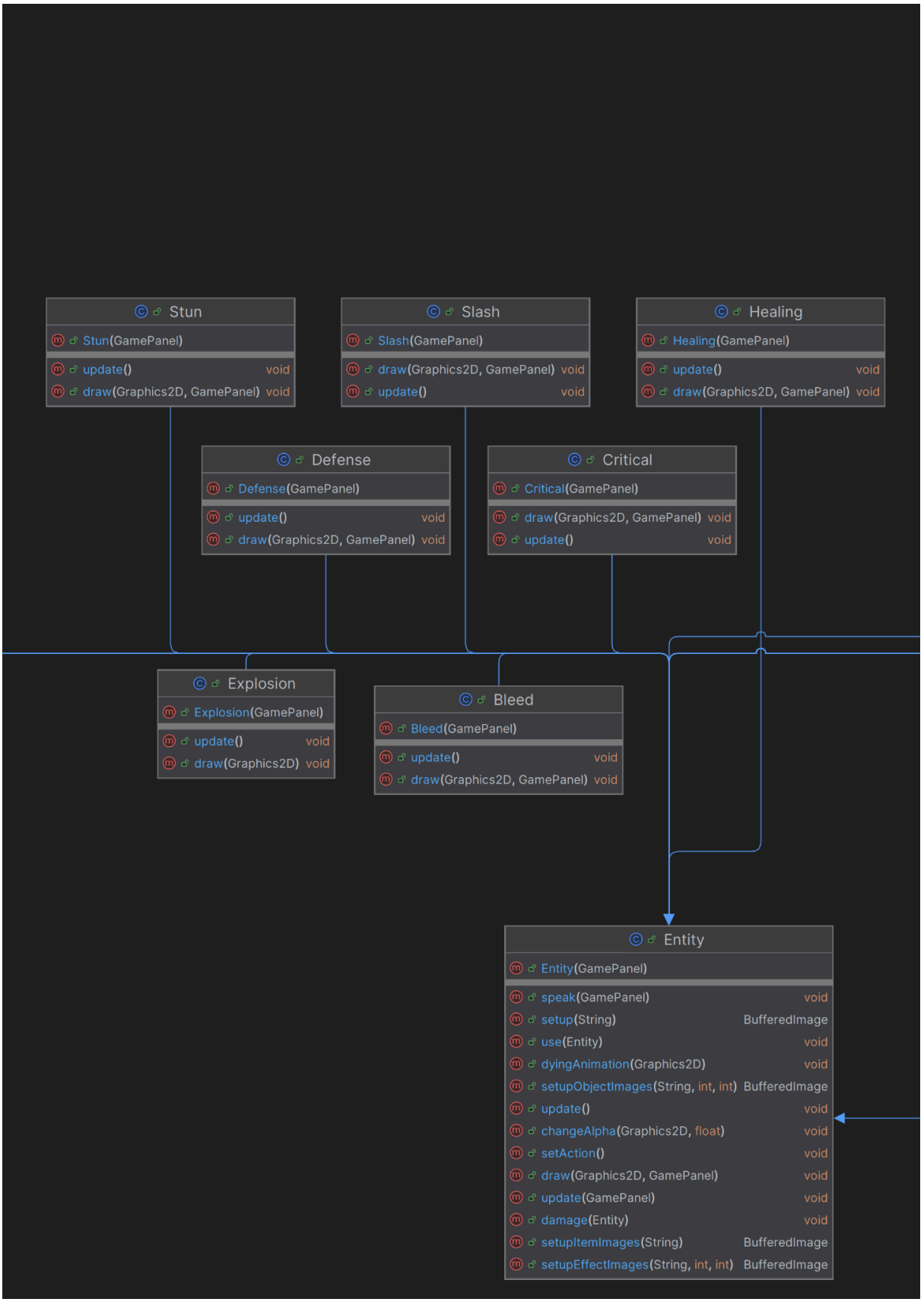
Monster diagram



Object diagram



Effect diagram



Chapter 3. DESIGN & IMPLEMENTATION

1. Programming Languages

Java: Primary languages for Game development project.

a. Project Structure

For this project, my group has decided to use IntelliJ IDEA and VSCode as an IDE to develop the application which members have frequently used, and excellent community support when they get stuck. This IDE and VSCode also provides tools like, Terminal, and Debugging Tool for testing and assurance purposes when tackling hard-to-achieve tasks, Tool for creating UML. In particular, IntelliJ IDEA and Visual Studio Code support adding external libraries easily.

We can summarize the folders and files' purpose as follows:

- **.idea:** for storing IDE-specific project configurations (JDK level, the pattern, the structure, and many more).
- **out:** the compiled code to run the application.
- **src:** the project's source code mainly contains the class and .fxml files used by G.U.I. Designer.
- **.gitgnore, README.md:** used for project setup on GitHub Version Control System and provide based configuration per developers.
- **target:** is the Maven build directory, which means that all generated content should be placed under that folder.
- config text to save the parameter using for option setting
- **pom.xml:** is where everything related to the project configured through maven is declared, such as declaring dependencies.

2. Class Structure

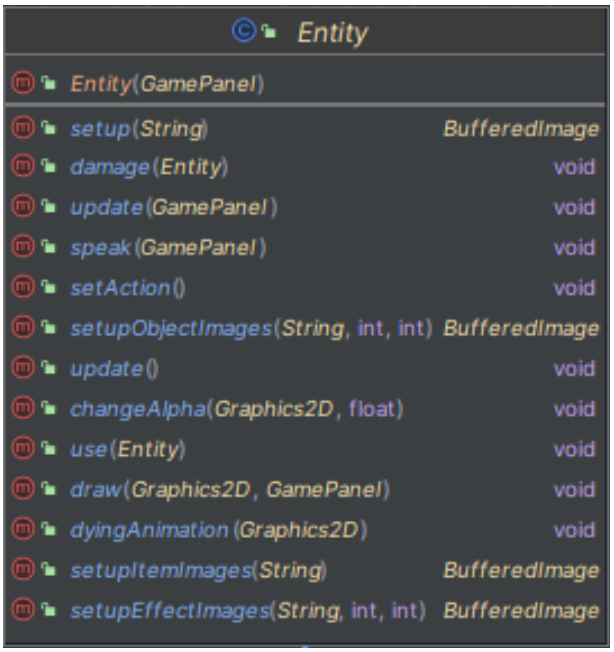
a. Main.java

- manages the main() function to launch and start the application process.

```java
package Main;

import ...


TrQAnhh +1 *
public class Main {

    1 usage
    public static Window window;

    TrQAnhh +1 *
    public static <image> void main(String[] args) {
    // GENERATING WINDOW JFRAME:
        JFrame window = new JFrame( title: "Finantasy");

    // WINDOW SETTINGS:

        // SET WINDOW ICON IMAGE
            ImageIcon imageIcon = new ImageIcon( filename: "res/Logo/logo.png");
            window.setIconImage(imageIcon.getImage());

        // GAME PANEL SETTING:
            window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            window.setResizable(false);


            GamePanel gamePanel = new GamePanel();
            window.add(gamePanel);

            gamePanel.config.loadConfig();
            if(gamePanel.fullScreenOn == true) {
                window.setUndecorated(true);
            }

            window.pack();

            window.setLocationRelativeTo(null);
            window.setVisible(true);
            gamePanel.setupGame();
            gamePanel.startGameThread();

```

b. Entity.java

- manages the player, monster attribute, npc and weapon.

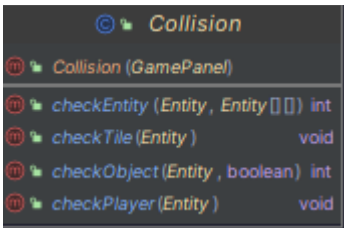


c. EventHandler.java

- manages the game event and start the application process fighting mode.

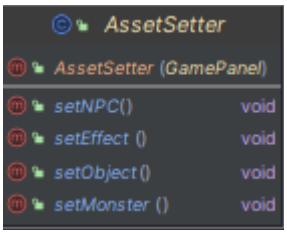
d. Collision.java

- manages the collision functions between each objects in the game.



e. AssetSetter.java

- manages the functions relative to set position for objects and effect fighting in game.



f. UtilityTool.java

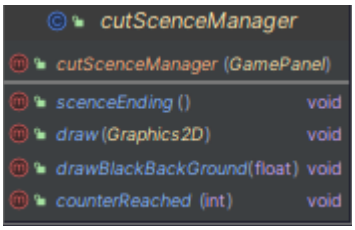
- class contain method for image scaling and processing.

g. EnviromentManager.java

- Processing lighting effect functions manage day and night event in games system.

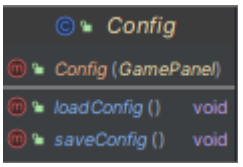
h. cutScenceManager.java

- Manage transfer state during playing state and end game state.



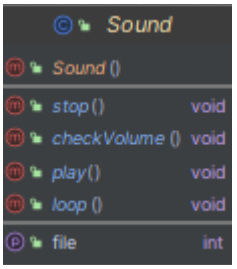
i. Config.java

- Contain methods handle saving and loading parameter of option setting .



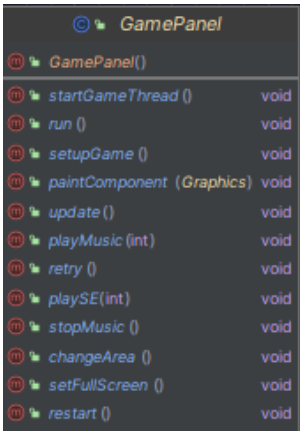
j. Sound.java

- Manage sounds effect, processing sound effect for game play



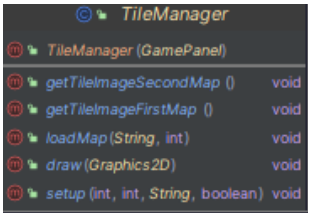
k. GamePanel.java

- Rendering (drawing the game's graphics, including backgrounds, characters, enemies, and other visual elements), Game Loop (running the main game loop, which includes updating game states and rendering frames at a regular interval), User Input Handling (capturing and processing user inputs, such as touch events), Game Logic (managing game states, checking for collisions, updating scores, and other game-specific logic), Resource Management (loading and managing game resources like images and sounds)



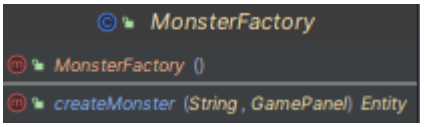
l. TileManager.java

- Loading Tile Data (loading tile images and map data from resources or files), Storing Tile Information (storing tile images and their properties), Rendering Tiles (drawing the tiles onto the game screen based on the player's position and the camera view), Tile Collision Detection (managing tile properties like walkability and handling collisions with tiles)



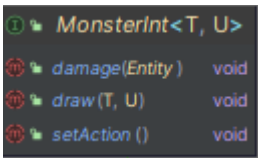
m. Monsterfactory.java

- Concrete class creation, override the factory method to return an instance of a concrete monster instance.



n. MosnterInt.Interface

- Defines the interface that all methods that concrete monsters must implement.

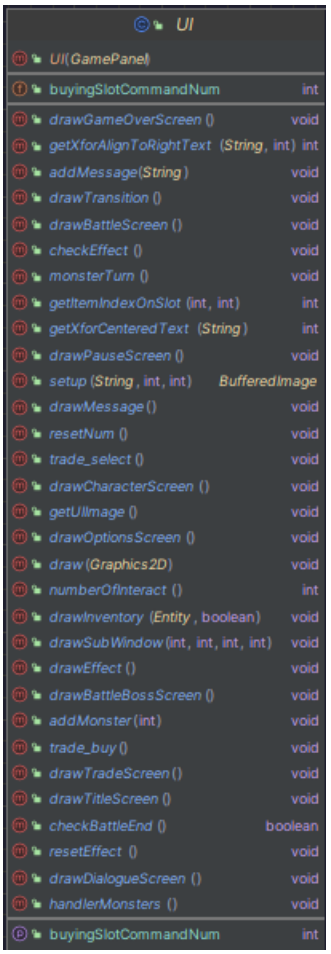


o. KeyHandler.java

- Detecting Key Presses (listening for key press events and determining which key was pressed), Handling Key Actions (mapping key presses to game actions, such as moving a character, pausing the game, or triggering an attack), Managing Key States for each of setting states in system (keeping track of the current state of keys (pressed or released)), GameBattle methods (Create and manage battle system panel for fighting mode)

p. UI.java

- Main Menu (options to start the game, load a game, access settings, and quit), In-Game HUD (Heads-Up Display) (displays essential game information like health, score, ammo, and mini-maps), Pause Menu (allows players to pause the game, resume, access settings, or quit to the main menu), Inventory System (enables players to view and manage their items, equipment, and resources), Dialog Boxes and Prompts (show messages, tutorials, and story elements), Settings Menu (allows players to adjust game settings like sound, graphics, controls, and gameplay options), Game Over and Victory Screens



3. UI Components

- The component-based architecture allows for a flexible and scalable approach to game design. Here are the main types of components being used in game system
  - ➢ Transform Component
  - ➢ Render Component
  - ➢ Physics Component
  - ➢ Input Component
  - ➢ Audio Component
  - ➢ AI Component
  - ➢ Animation Component
  - ➢ UI Component
  - ➢ Networking Component
  - ➢ Gameplay Component
  - ➢ Special Effects Component
  - ➢ Event System Component
  - ➢ Script Component
  - ➢ Resource Management Component
  - ➢ Saving and Loading Component

4. Specific UI component design

a. Main Menu

➢ **Start Game:** Button to begin the game.

➢ **Options:** Access to settings like audio, controls, graphics, and language.

➢ **Credits:** Information about the game's developers and contributors.

➢ **Exit:** Option to close the game.

b. In-Game HUD (Heads-Up Display)

➢ **Inventory:** A quick view of items the player has collected.

➢ **Objectives/Quests:** Displays current objectives or quests.

c. **Pause Menu**

➢ **Music:** Music volume setting

➢ **SFX:** Sound effect volume setting

➢ **Control:** Access to keyboards parameter settings.

➢ **Back:** To continue playing.

➢ **Exit Game:** Option to quit the game.

d. **Dialog Boxes/Notifications**

➢ **Pop-ups:** For in-game alerts, monster damage, or important notifications.

➢ **Dialogue System:** For conversations with NPCs (Non-Player Characters).

e. **Character Screen**

➢ **Description:** Displays the weapon's information.

➢ **Equipment:** Shows equipped items and allows changing equipment.

➢ **Player's parameter:** Displays player's level, money and other attributes.

f. **Game over Screen**

➢ **Retry:** To play again.

➢ **Exit:** Options to exit game.

g. **Trade/ buy shop screen**

➢ **Description:** Displays the weapon's information.

➢ **Equipment:** Shows equipped items and allows changing equipment.

h. **Battle monster screen**

➢ **Monsters information:** Displays the monster's information (name, HP and monster turn).

➢ **Options:** Access the player's select (Attack, Defend, Items) or select weapon, select monster to attack during the monster fight

➢ **Player information:** Displays the player's information (name, HP and player turn).

i. **End game screen**

➢ Credit: Display the information of developer's team

5. Game-Specific Components

- Game Loop: The main loop that handles updating and rendering the game.
- Game Engine: Custom or third-party libraries for handling game physics, rendering, etc.
- Sprites and Textures: Graphics for characters, environments, etc.
- Sound and Music: Audio files and playback handling.

6. Design Pattern

- Factory design pattern: Our team uses the factory design pattern for monsters class to provide a way to create monster objects without specifying the exact class of object that will be created. This pattern is particularly useful when the exact type of object to be created isn't known until runtime or when the creation process is complex or involves multiple steps. The Factory Pattern promotes loose coupling and enhances code maintainability and scalability.

```java
package Monster;

import Entities.Entity;

public interface MonsterInt<T, U> {
    public void setAction();
    public void damage(Entity entity);
    public void draw(T t, U u);
}
```

```java
package Monster;

import Entities.Entity;
import Main.GamePanel;

public class MonsterFactory{
    public static Entity createMonster(String monsterType, GamePanel gamePanel) {
        switch (monsterType) {
            case "GhostRider":
                return new MON_GhostRider(gamePanel);
            case "Pumpkin":
                return new MON_Pumpkin(gamePanel);
            case "Reaper":
                return new MON_Reaper(gamePanel);
            case "Spider":
                return new MON_Spider(gamePanel);
            case "Gate Keeper":
                return new MON_GateKeeper(gamePanel);
            case "Bloody Slime":
                return new MON_BloodySlime(gamePanel);
            case "Robot":
                return new MON_Robot(gamePanel);
            case "Green Dragon":
                return new MON_GreenDragon(gamePanel);
            case "Red Pheonix":
                return new MON_RedPheonix(gamePanel);
            case "Boss":
                return new MON_Boss(gamePanel);
            default:
                throw new IllegalArgumentException("Unknown monster type: " + monsterType);
        }
    }
}
```

- Singleton Pattern ensures that a class has only one instance and provides a global point of access to it. This is useful for our team managing shared resources like configurations, loggers player

```
public class Player extends Entity{
    private static Player instancePlayer = null;
    // VARIABLES:
        public final int screenX;

        public final int screenY;
        KeyHandler keyHandler;
    // CONSTRUCTORS:
        public Player(GamePanel gamePanel, KeyHandler keyHandler){ ···
        public static Player getInstance(GamePanel gamePanel, KeyHandler keyHandler) {
            if(instancePlayer == null) {
                return new Player(gamePanel, keyHandler);
            }
            return instancePlayer;
        }
```

```
// Player CLASS
    public Player player = Player.getInstance(this, keyHandler);
// TileManager CLASS
```

Chapter 4. FINAL APP GAME

Source code (link Github):

https://github.com/TrQAnhh/Finantasy

Gameplay:

1. Title Screen

We add the sound for the Title Screen.

You can select:

- **Play:** Start the game.
- **Setting:** You can read the game tutorial and adjust the volume of music and sfx.

- **Control:** show your default settings from keyboards
- **Exit:** Exit the game.

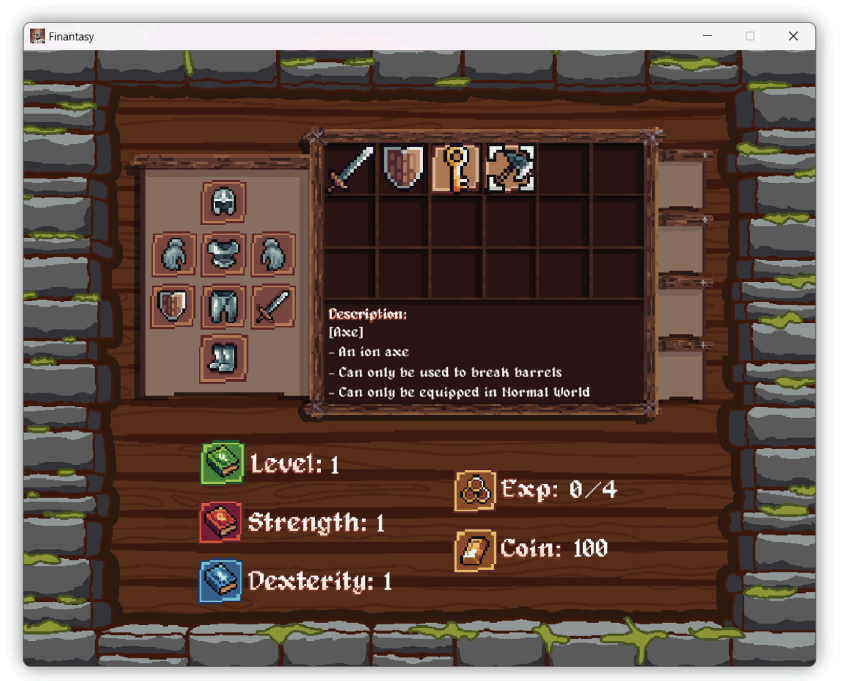
2. Gameplay Screen

The beginning:

Interact with the NPC: Talk to the NPC to follow the story path.

Open inventory: by pressing “C” button.

In the inventory screen, you can see your character attributes



including current level, strength, dexterity, experience (Exp) and coin.

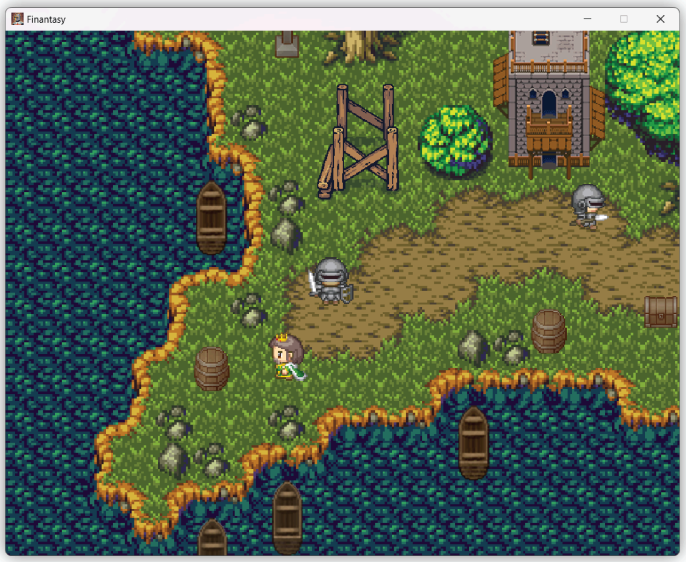
Interact with objects: You can interact with objects to get rewards.

- **Step 1:** hitting the chest

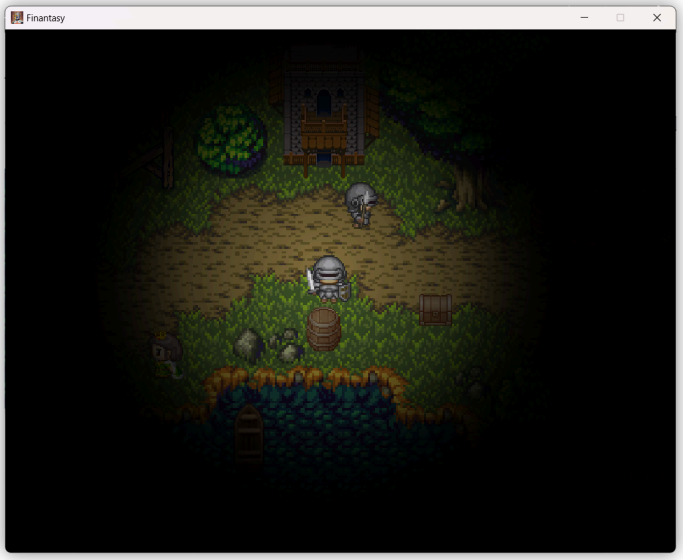
- **Step 2:** equipping the key (or the weapons that is enable to used with suitable objects)

- **Step 3:** pressing enter to interact

Day and Night: We set a Night mode to make the game scarier.



Day mode



Night mode

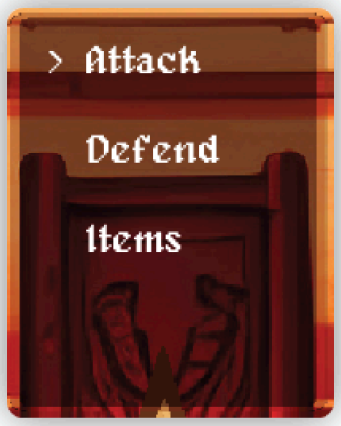
3. Battle Screen

How to fight: You need to stand on specific points

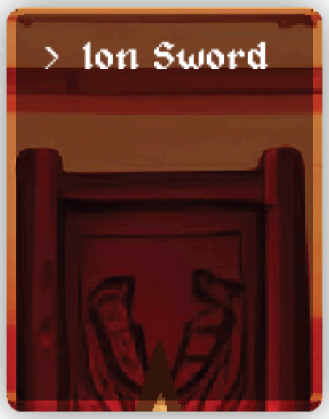
We also add the sound for the Battle State.

Inside the battle, you can interact to choose your action. We create 3 section include:

- **Select Action**: You can choose your action below:
  - **Attack:** Attack the enemy.



  - **Defense:** Defense your player.
  - **Items:** Use a special item in your inventory.
- **Choosing Attack Equipment**: You can choose your equipments, items in your inventory



- **Choosing Entity**: You can choose your enemy to attack, or your player to defend, or using items.

After your selection, it will run the effect and damage:
Now, you can play until you or all enemies die.

If you die, you can choose Retry the game or Exit the game:

After a fight, you can interact with the holy statue to heal your HP.

4. Trade Screen

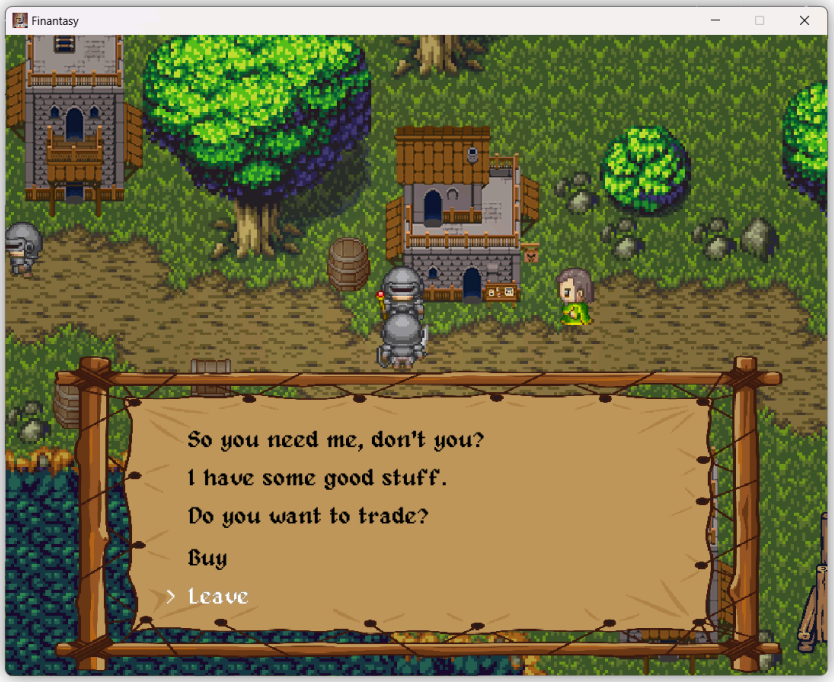
When you kill monsters, it will drop an amount of gold for you to buy more powerful equipment and items.

In this screen, you can see the attributes and information of items. You can press the “ENTER” button to buy the item.

If you do not have enough gold, the NPC will warn you

You can press “ESC” button to return the dialogue

And choose Leave when your buying is finished.



5. Cave

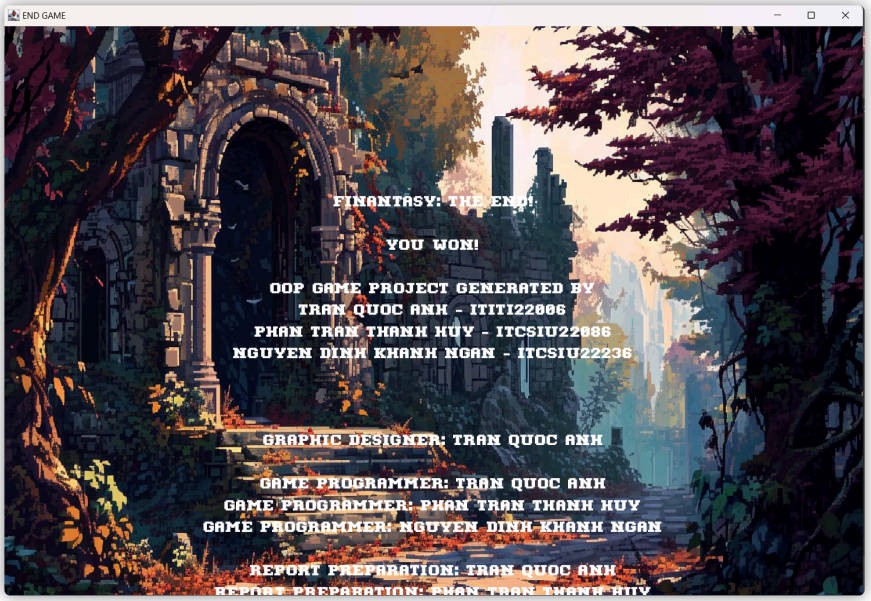
There is a cave where you can enter it

You can hear the sound inside the cave.
Many powerful monsters are inside the cave, including the final boss.

So your aim is to kill as many monsters as possible to earn gold, buy powerful items to deal with monsters until you kill the boss, and complete the story.

6. Credit

- After defeating the final dragon boss, you'll be able to view the credits showcasing all our team's contributions and contact information for each teammate.



Chapter 5. EXPERIENCE

During the time working on the project, our team comes to understand that the software component of the game is just one of many components that need to be taken into account in relation to the game's environment, plot, gameplay, artwork, animation, and so forth.

We have the opportunity to practice many of the lessons we have learned in class and cope with a variety of bugs. Additionally, it motivates pupils to study more on their own time. As a result, we not only solidify the lessons learned, but also get new experiences and keep up with technological advancements.

At last, our group understands that our major requires students to do most of their own independent work. The IT world is too vast to wait and rely just on classroom instruction.

My team will try our best to complete this game as the first project to publish it as soon as we can and improve it to become more exquisite, superior, and varied.