VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

School of Computer Science & Engineering

OBJECT - ORIENTED PROGRAMMING PROJECT

Topic: Platform Game - “Mysterious Journey”

Instructor: Tran Thanh Tung

Members:
Nguyen Phuc Dat - ITITIU22028
Tran Luu Hong Phuong - ITITIU22198
Nguyen Hoang Thao Trinh - ITCSIU22288
Nguyen Thanh Nam ITCSIU22311

TABLE OF CONTENTS

I. INTRODUCTION

1. Objectives

The game is based on the basic concept of the iconic platform game "Mario", but instead of taking the same assets as the original game, our group has changed it into something dark, mysterious, and witchy to fit the name "Mysterious Journey". In fact, we've learnt to make this game through instruction from the Youtuber "Kaarin Gaming" and also added some additional changes to make it our own game.

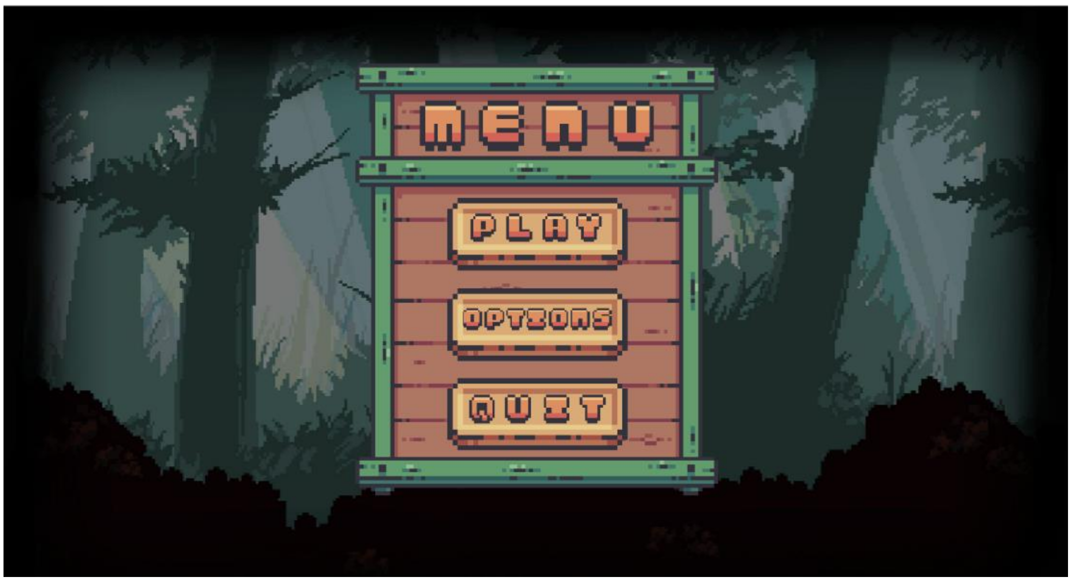


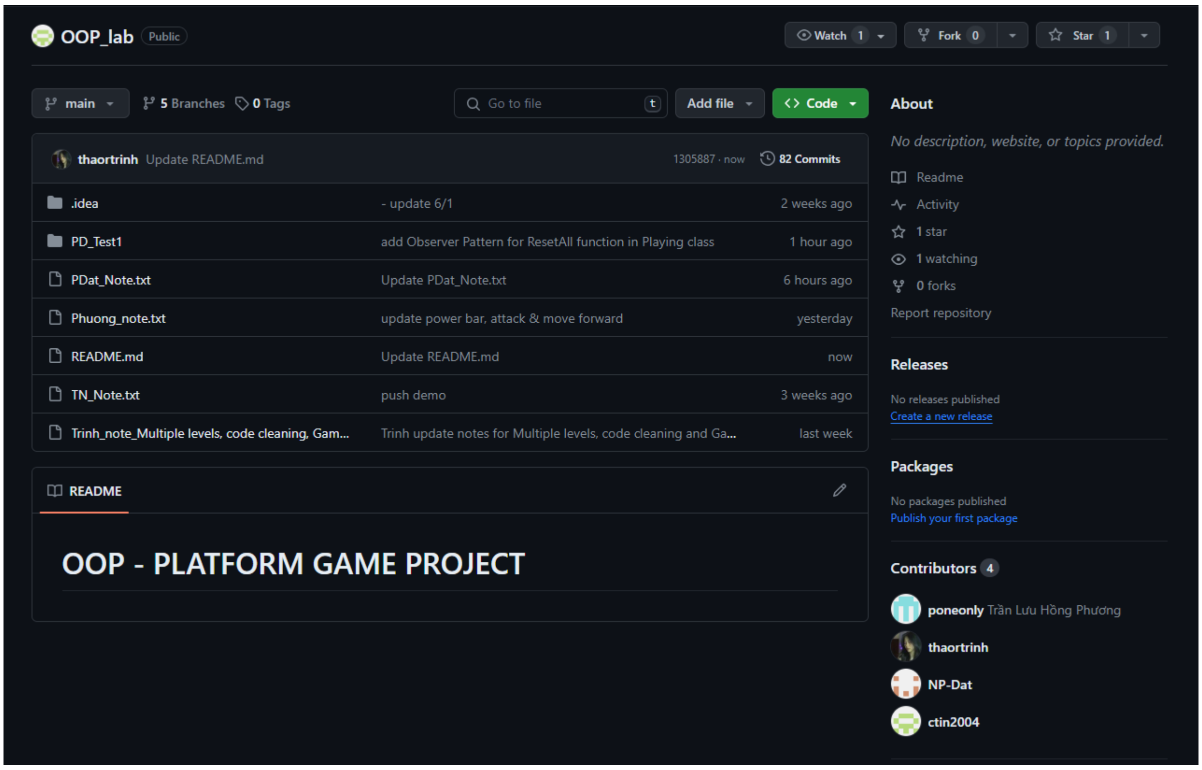
Figure 1.1: Game Menu

This game project aims to:

- Create a platform game for entertaining purposes.
- Practicing some fundamental concepts of Object - Oriented Programming
- Develop teamwork and problem-solving skills during the project.

2. The tool used

+ **IDEs for programming and debugging:** JetBrains IntelliJ IDEA and Visual Studio Code
+ **Code version management:** Github
+ **Project Management:** Github
+ **Communication and Weekly meetings:** Discord


OOP_lab Public

Watch 1 · Fork 0 · Star 1

main · 5 Branches · 0 Tags · Go to file · Add file · Code

thaortrinh Update README.md · 1305887 · now · 82 Commits

.idea	- update 6/1	2 weeks ago
PD_Test1	add Observer Pattern for ResetAll function in Playing class	1 hour ago
PDat_Note.txt	Update PDat_Note.txt	6 hours ago
Phuong_note.txt	update power bar, attack & move forward	yesterday
README.md	Update README.md	now
TN_Note.txt	push demo	3 weeks ago
Trinh_note_Multiple levels, code cleaning, Gam...	Trinh update notes for Multiple levels, code cleaning and Ga...	last week

README

OOP - PLATFORM GAME PROJECT

About

No description, website, or topics provided.

Readme · Activity · 1 star · 1 watching · 0 forks · Report repository

Releases

No releases published
Create a new release

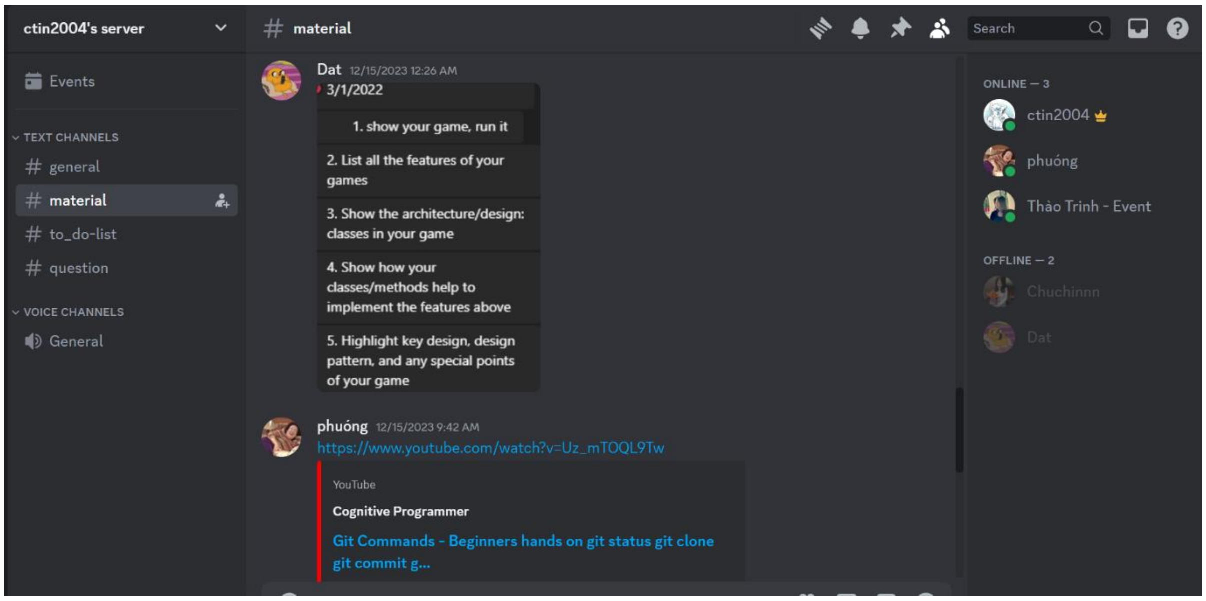
Packages

No packages published
Publish your first package

Contributors 4

poneonly Trần Lưu Hồng Phương
thaortrinh
NP-Dat
ctin2004

Figure 1.2: Github - Code version management


ctin2004's server · # material

Events

TEXT CHANNELS
# general
# material
# to_do-list
# question

VOICE CHANNELS
General

Dat 12/15/2023 12:26 AM
3/1/2022
1. show your game, run it
2. List all the features of your games
3. Show the architecture/design: classes in your game
4. Show how your classes/methods help to implement the features above
5. Highlight key design, design pattern, and any special points of your game

phuóng 12/15/2023 9:42 AM
https://www.youtube.com/watch?v=Uz_mTOQL9Tw
YouTube
Cognitive Programmer
Git Commands - Beginners hands on git status git clone git commit g...

ONLINE — 3
ctin2004
phuóng
Thảo Trinh - Event

OFFLINE — 2
Chuchinnn
Dat

Figure 1.3: Discord

II. GAMEPLAY & GAME RULES

1. Gameplay

Our team created a type of game called 'platformer' - a platform game where the player can control the character to go through an environment with different actions, such as running, jumping and attacking the enemies.

Our team implemented one of the platform game's main features, which is "Controlled jumping". It means that the player can control the character to jump with gravity so that the character can avoid the attack from the enemy or the bullets.

2. Game rules

The player must complete the levels by defeating all the enemies (the Crabs) while avoiding the obstacles and collecting the items to upgrade the character's status.

Control: the player controls the character's movements by pressing the "A" and "D" keys, which correspond to running left and right, respectively; pressing the "space" bar to jump; clicking the left mouse button to attack and right mouse to attack and move forward for a certain distance at the same time.

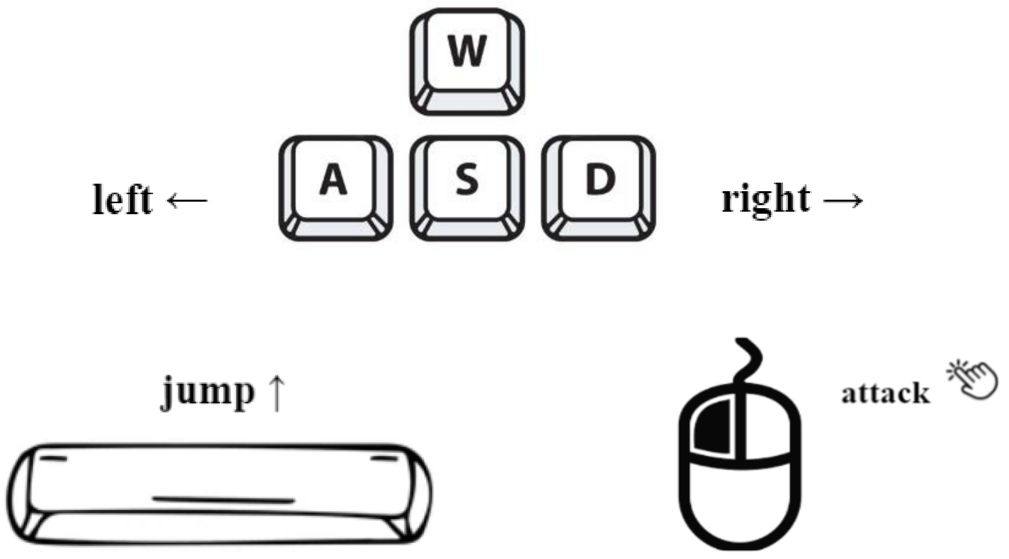


Figure 2.1: Control input

Levels: our game is divided into 3 levels, with an increased number of enemies and difficulty after passing one level.

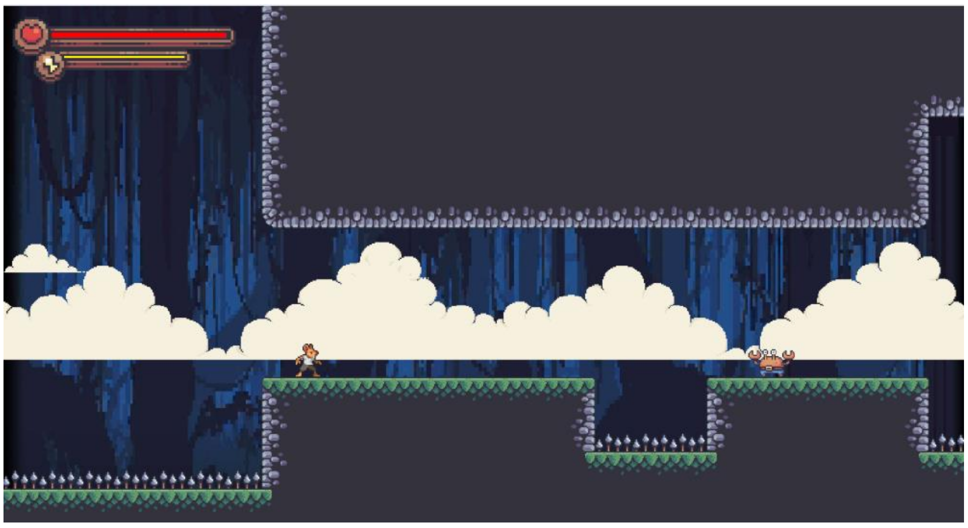


Figure 2.2: Level 1

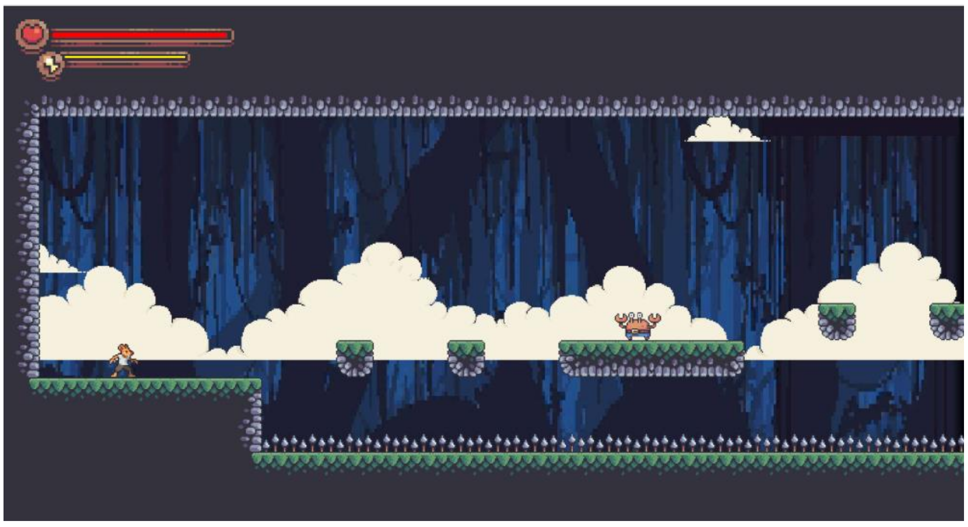


Figure 2.3: Level 2

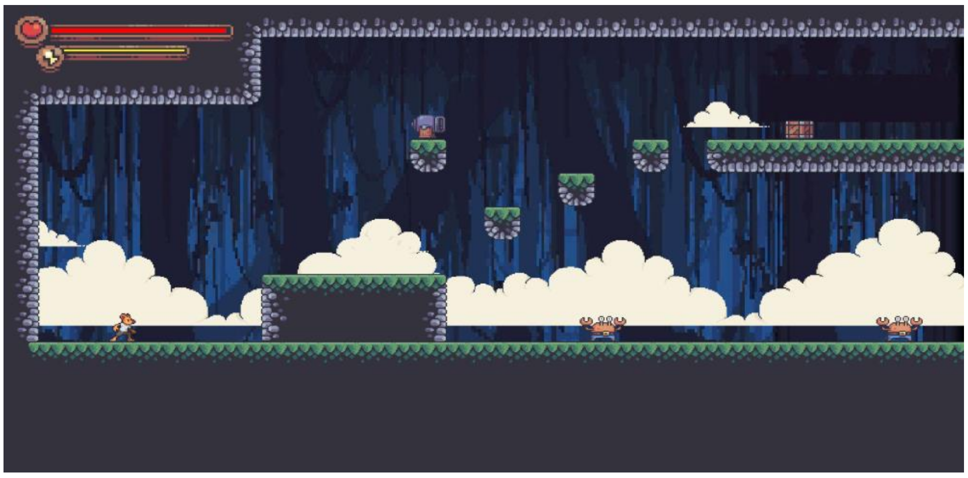


Figure 2.4: Level 3

Obstacles and Enemies: the levels are filled with obstacles like Crabs, traps and canons. The player must find and defeat all the enemies existing on that level by clicking the mouse to attack and kill the crabs, meanwhile avoiding the attacks from bullets of the canons, and avoiding the traps which can kill the character once touched.

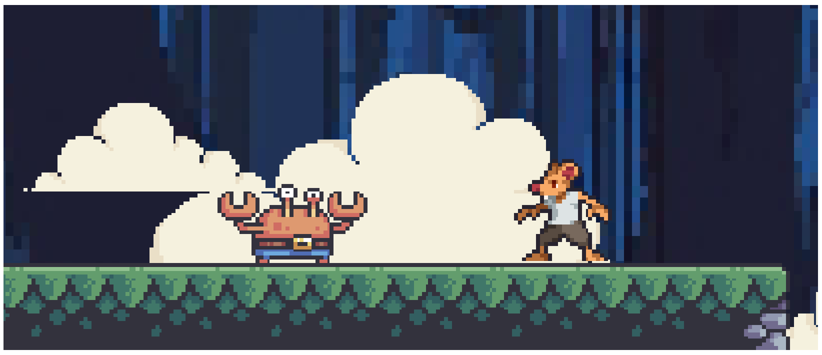


Figure 2.5: Crabs

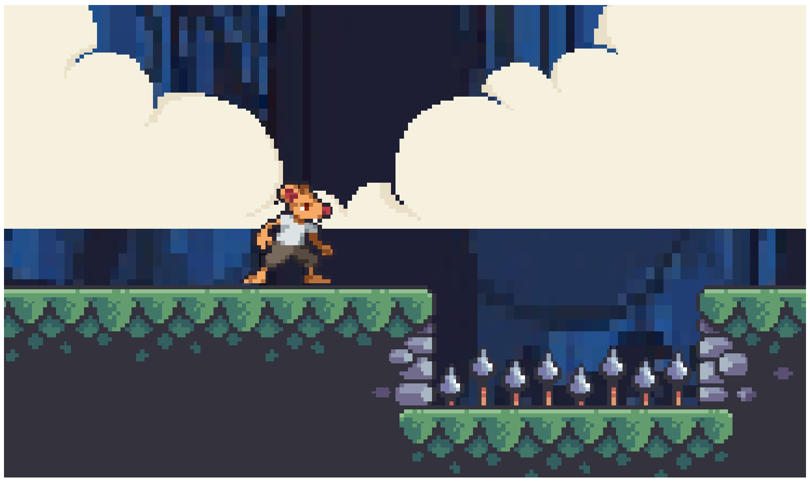


Figure 2.6: Traps

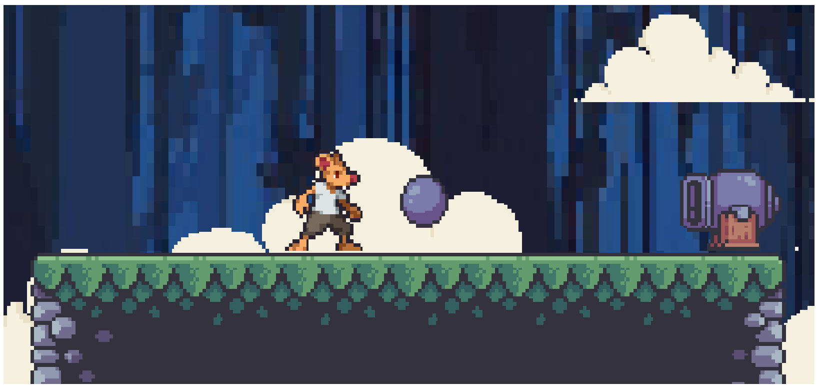


Figure 2.7: Canons

Power-ups: the player can collect power-ups that enhance the character's abilities. Common power-ups include the Red Potion (upgrade health) and Blue Potion (upgrade attack power)

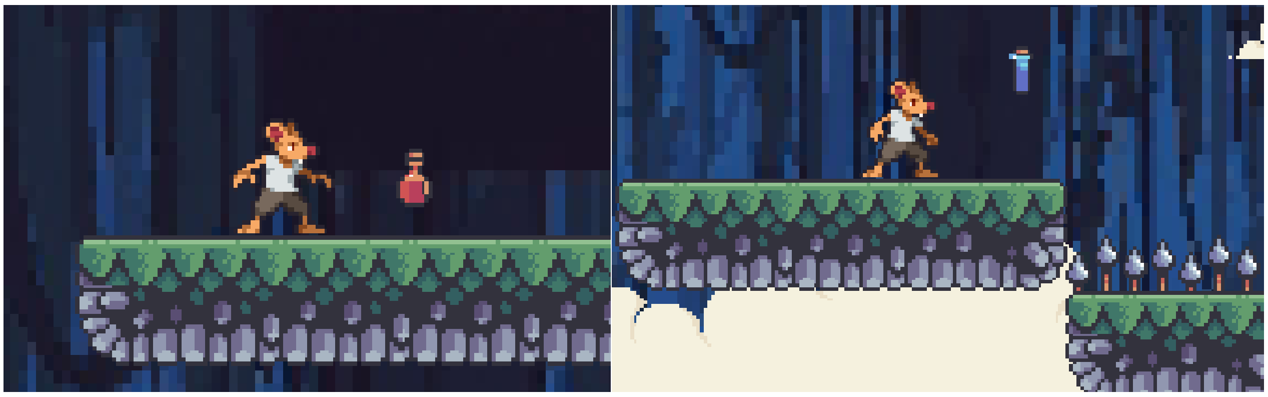


Figure 2.8: Power-up potions

Power-bar: each time the player attacks and runs at the same time, the energy will be minus 60 compared to the maximum energy of 200. The player can stand still or run without attacking to recover energy.

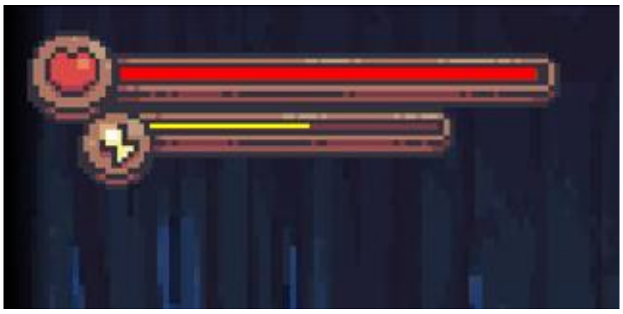


Figure 2.9: Power bar & Health bar

III. DETAILS

1. Functionality:

a. **Package "main":** contains 4 main classes:

- MainClass class: holds the main method to run the game.
- Game class:
  + combines everything such as handlers, levels, player, enemies,...; contains the main elements from other classes to build the game (render, update, run,..).
  + holds our game loop.
- GameWindow class (JFrame): used to make GUI (Graphical User Interface).
- GamePanel class (JPanel): draw, render the game scenes; set size to fit the window.

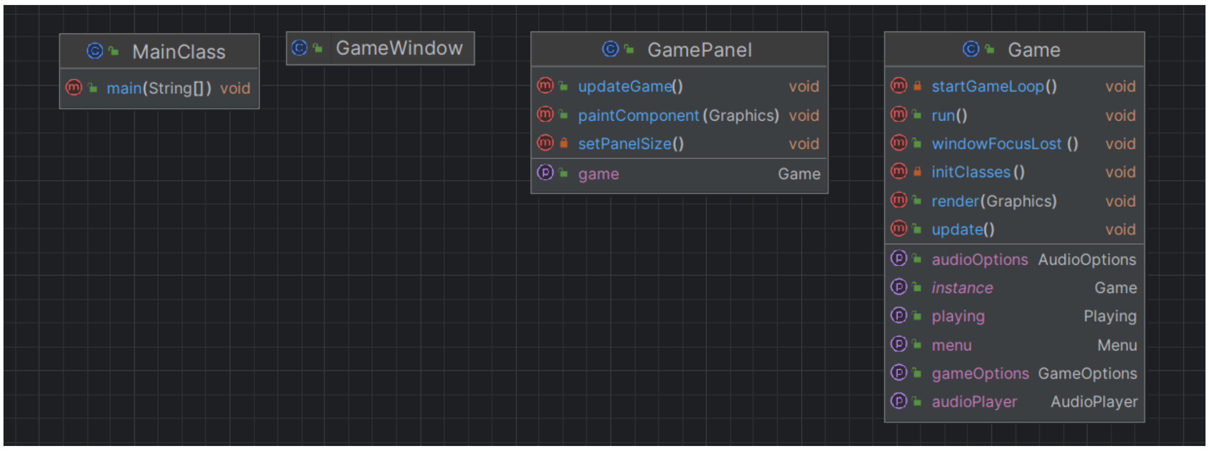


Figure 3.1: Package ”main”

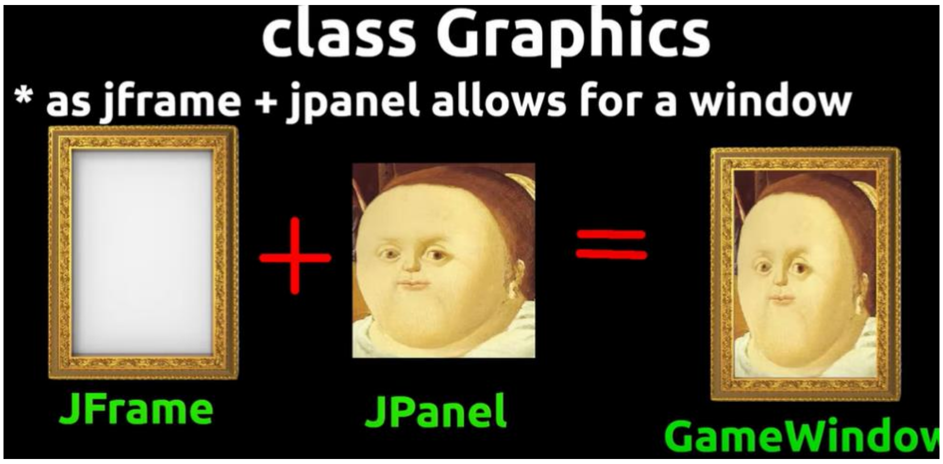
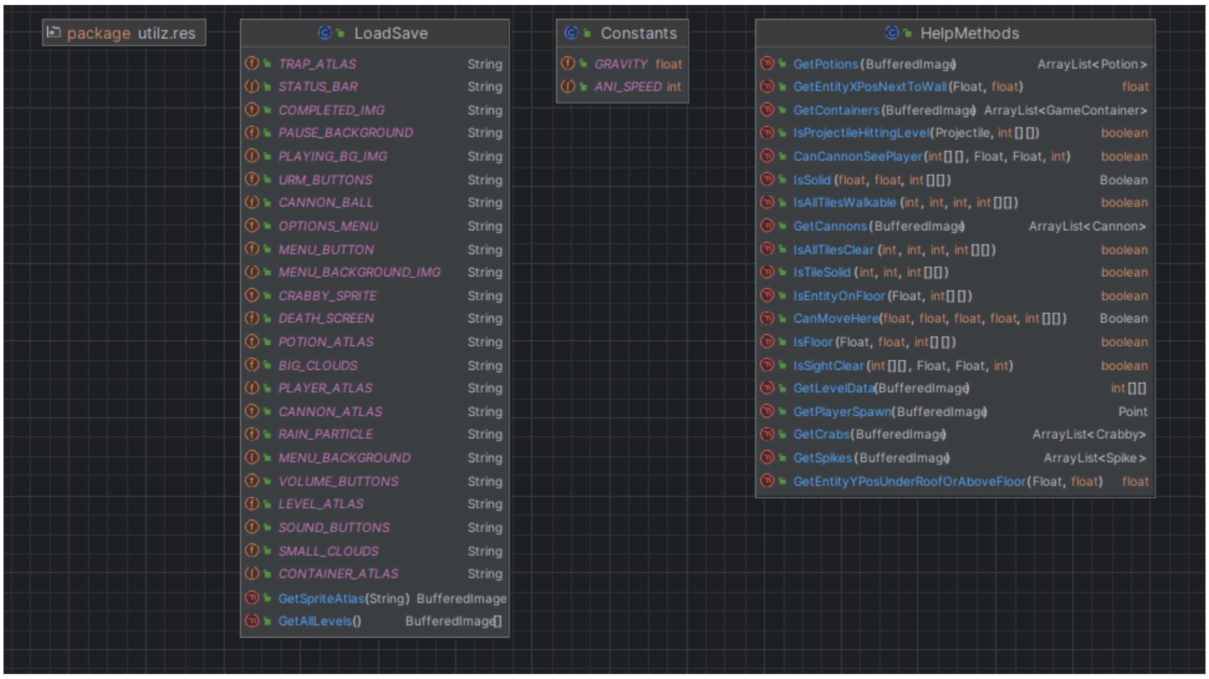


Figure 3.2: Game Window’s explanation

b. **Package “utilz”:**

- file res: to store the images used in the game, such as player sprite, background, and buttons,...
- have classes to contain default data (width, height, values of objects), store the address of the picture, some methods to read the image.



```
LoadSave
TRAP_ATLAS                String
STATUS_BAR                String
COMPLETED_IMG             String
PAUSE_BACKGROUND          String
PLAYING_BG_IMG            String
URM_BUTTONS               String
CANNON_BALL               String
OPTIONS_MENU              String
MENU_BUTTON               String
MENU_BACKGROUND_IMG       String
CRABBY_SPRITE             String
DEATH_SCREEN              String
POTION_ATLAS              String
BIG_CLOUDS                String
PLAYER_ATLAS              String
CANNON_ATLAS              String
RAIN_PARTICLE             String
MENU_BACKGROUND           String
VOLUME_BUTTONS            String
LEVEL_ATLAS               String
SOUND_BUTTONS             String
SMALL_CLOUDS              String
CONTAINER_ATLAS           String
GetSpriteAtlas(String) BufferedImage
GetAllLevels()         BufferedImage[]
```

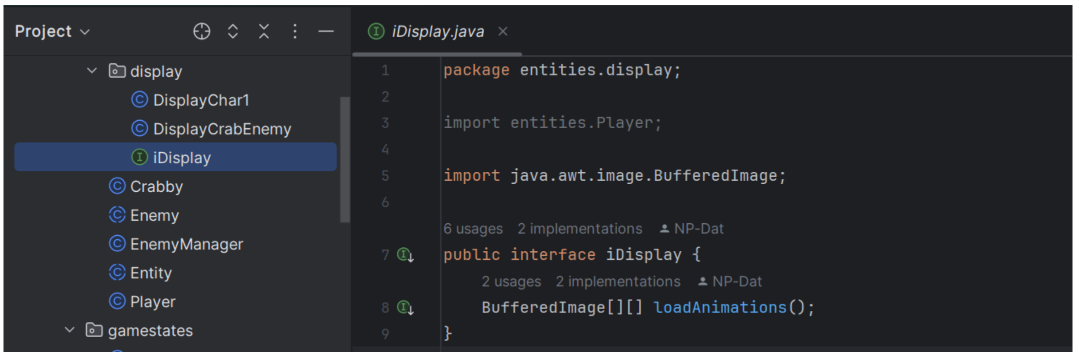
```
Constants
GRAVITY   float
ANI_SPEED int
```

```
HelpMethods
GetPotions(BufferedImage)                            ArrayList<Potion>
GetEntityXPosNextToWall(Float, float)                float
GetContainers(BufferedImage)                         ArrayList<GameContainer>
IsProjectileHittingLevel(Projectile, int[][])        boolean
CanCannonSeePlayer(int[][], Float, Float, int)       boolean
IsSolid (float, float, int[][])                      Boolean
IsAllTilesWalkable (int, int, int, int[][])          boolean
GetCannons(BufferedImage)                            ArrayList<Cannon>
IsAllTilesClear (int, int, int, int[][])             boolean
IsTileSolid (int, int, int[][])                      boolean
IsEntityOnFloor(Float, int[][])                      boolean
CanMoveHere(float, float, float, float, int[][])     Boolean
IsFloor (Float, float, int[][])                      boolean
IsSightClear(int[][], Float, Float, int)             boolean
GetLevelData(BufferedImage)                          int[][]
GetPlayerSpawn(BufferedImage)                        Point
GetCrabs(BufferedImage)                              ArrayList<Crabby>
GetSpikes(BufferedImage)                             ArrayList<Spike>
GetEntityYPosUnderRoofOrAboveFloor(Float, float)     float
```

Figure 3.3: Package ”utilz”

c. **Package “entities”:**

- Containing many subclasses to control many characters like “Player”, and “Enemy” which inherits from the class “Entity”; enemies are controlled by the “EnemyManager” class.
- The sub-package "display", an interface iDisplay is designed based on SOLID – Open/Closed principle, it is opened for extension so that both Player (DisplayChar1) and Enemies (DisplayCrabEnemy) or more characters can implement this interface to display their own animations.
-



```java
package entities.display;

import entities.Player;

import java.awt.image.BufferedImage;

public interface iDisplay {
    BufferedImage[][] loadAnimations();
}
```

Figure 3.4: iDisplay interface

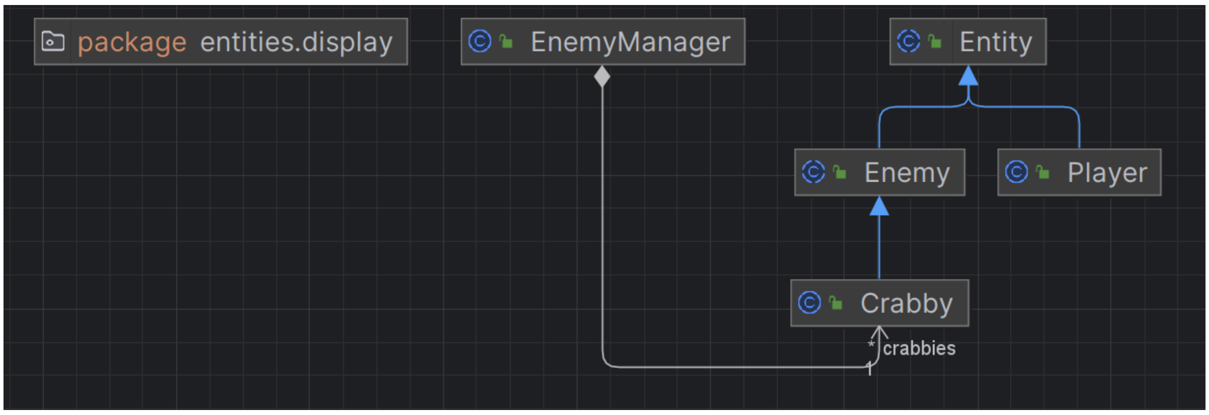


Figure 3.5: Package ”entities”

d. **Package “objects”:**

- has the “GameObject” class as a base and some subclasses inherit the base like the “Cannon”, “Spike”, “Potion” and “GameContainer” classes; some independent classes such as “Projectile” and “ObjectManager” that manages the behavior of these objects.

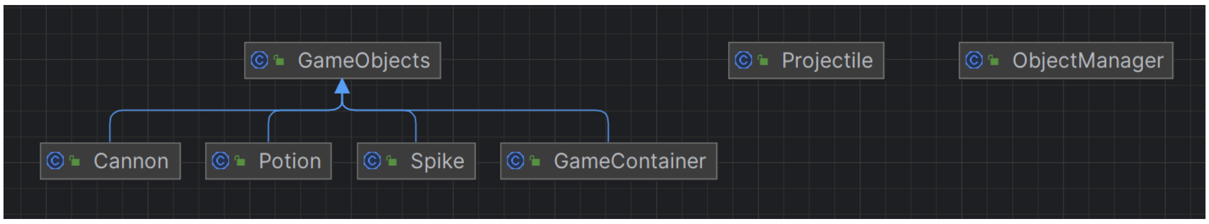


Figure 3.6: Package ”objects”

e. **Package “levels”:**

- manage the levels of the game, create necessary things in each level (potions, enemies, playerSpawn, game objects...).
- LevelManager class to manage the current level and load next level.

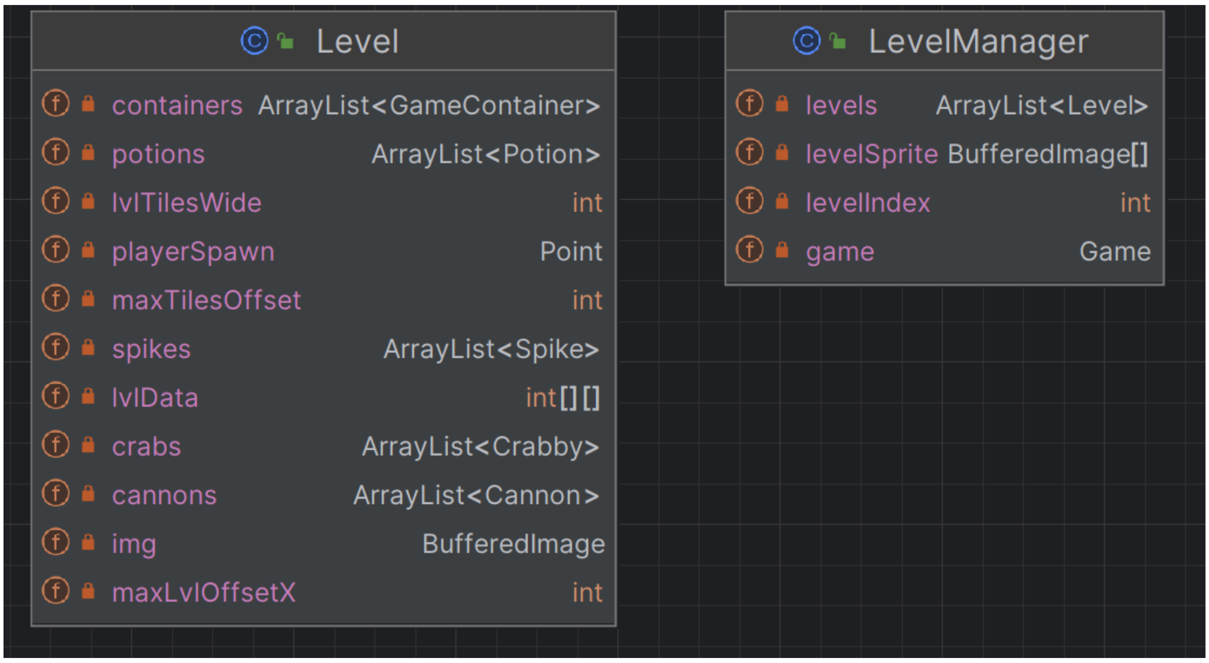


Figure 3.7: Package ”levels”

f. Package “gamestates”:

– manage the states of the game (menu or playing).

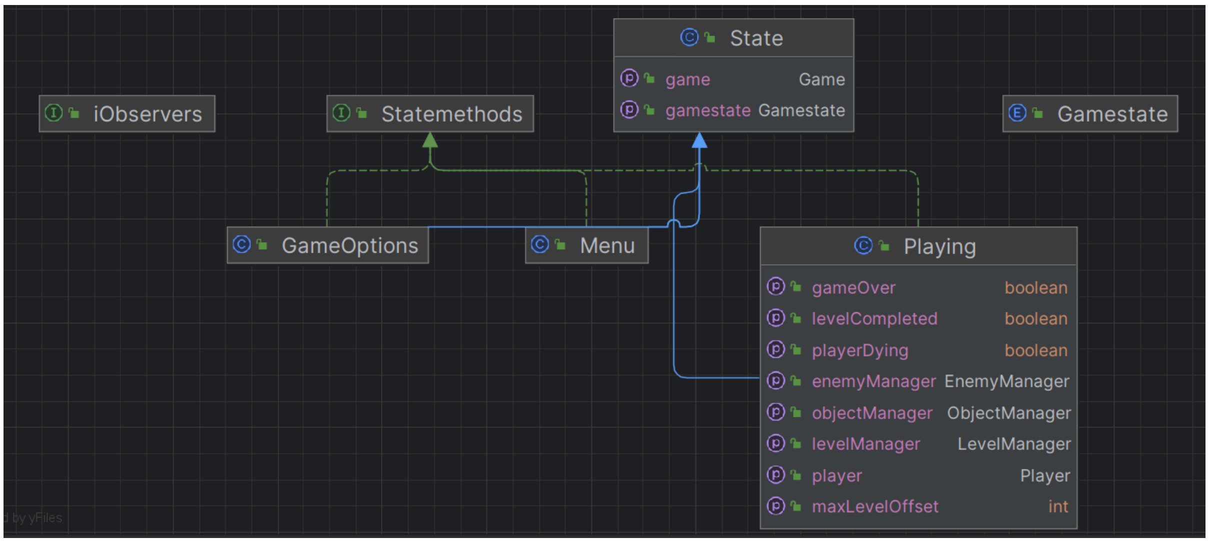


Figure 3.8: Package “gamestates”

g. **Other packages:**

- **ui:** manage the GUI (display on the screen the selection buttons like sound, pause, audio, volume,...).

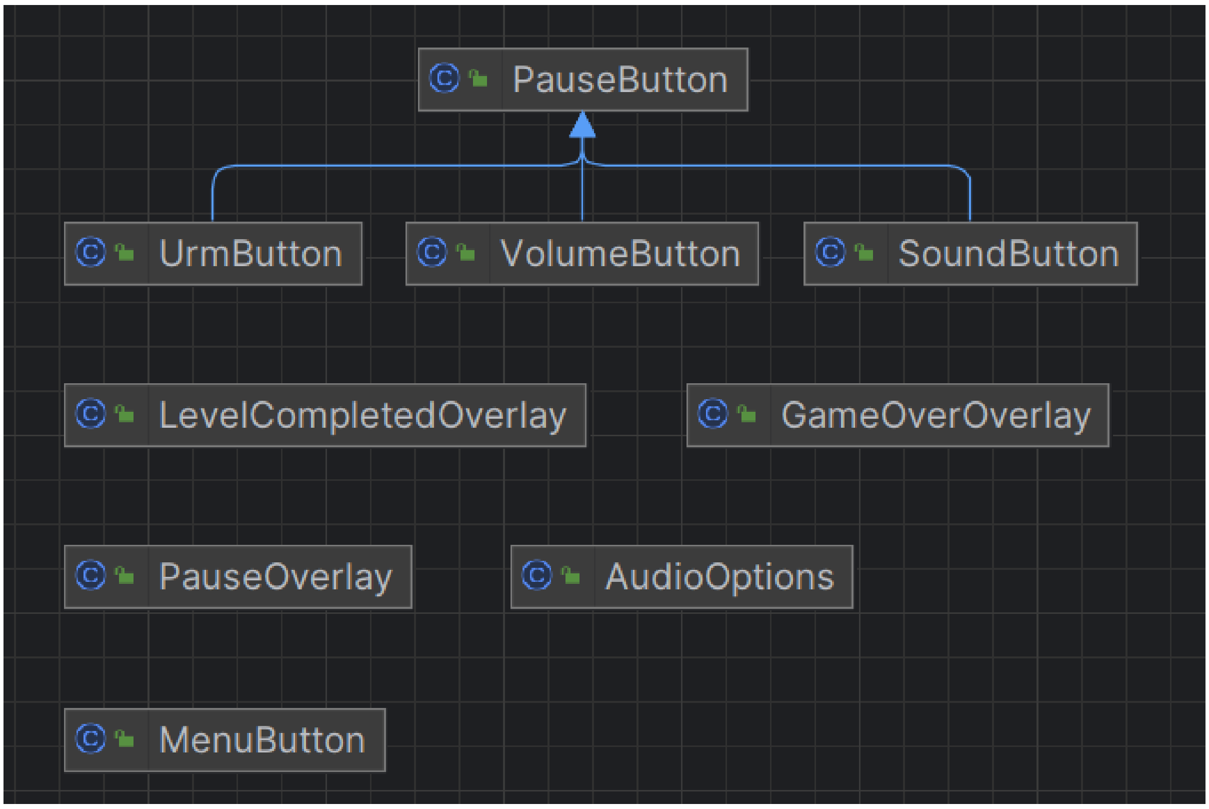


Figure 3.9: Package ”ui”

- **inputs:** manage the way game receives the signal from keyboard and mouse.

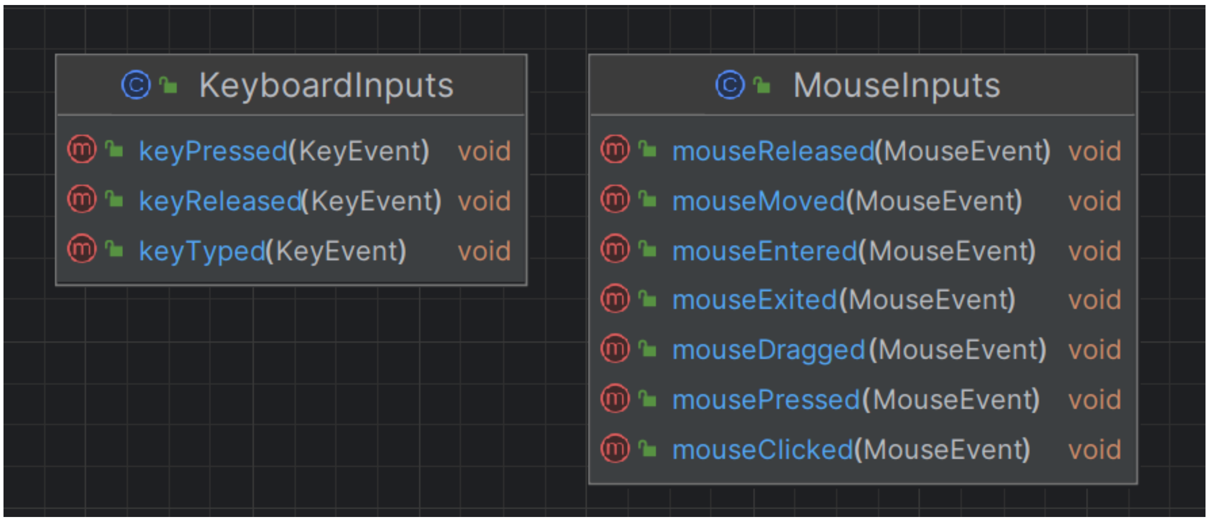

KeyboardInputs
keyPressed(KeyEvent) void
keyReleased(KeyEvent) void
keyTyped(KeyEvent) void
MouseInputs
mouseReleased(MouseEvent) void
mouseMoved(MouseEvent) void
mouseEntered(MouseEvent) void
mouseExited(MouseEvent) void
mouseDragged(MouseEvent) void
mousePressed(MouseEvent) void
mouseClicked(MouseEvent) void

Figure 3.10: Package ”input”

- **audio:**

+ “AudioPlayer” class: plays sound for the game, switches music in each level, specific sound for certain events(death, attack,...)

- **file audio**: to store the audio used in game such as attack sounds, jump sound, die sound, etc

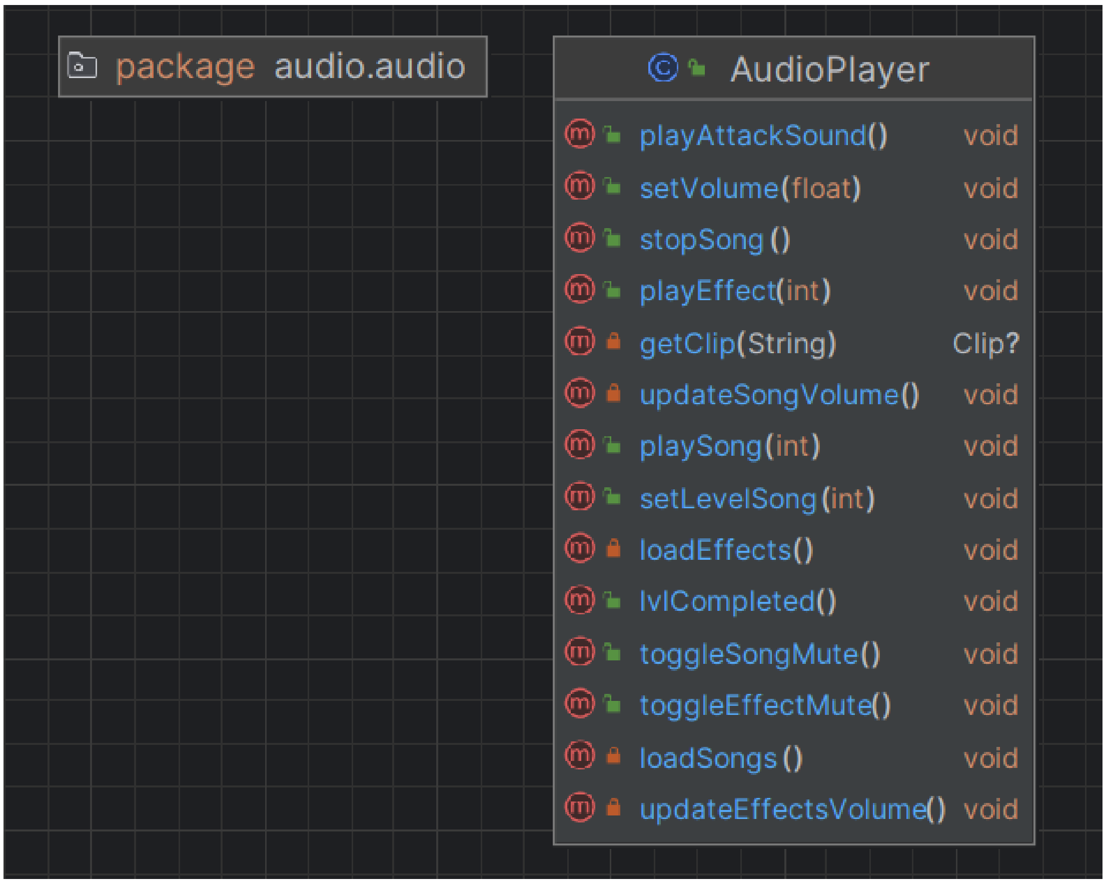


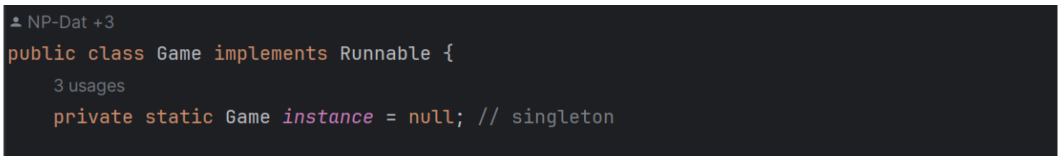
Figure 3.11: Package ”audio”

2. Applied design patterns:

Our team have applied some concepts of design patterns:

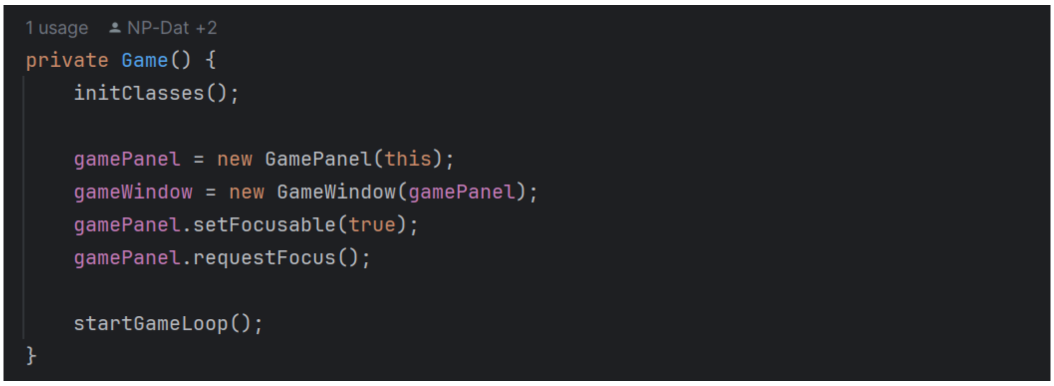
- **Singleton**

- In Game class



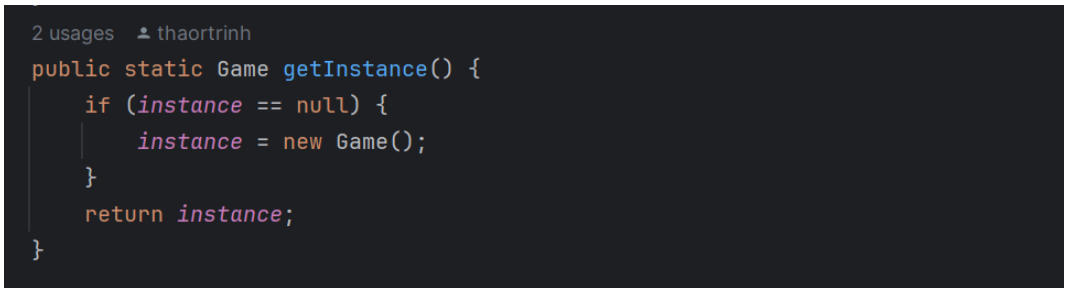
```
NP-Dat +3
public class Game implements Runnable {
    3 usages
    private static Game instance = null; // singleton
```

Figure 3.12



```
1 usage  NP-Dat +2
private Game() {
    initClasses();

    gamePanel = new GamePanel(this);
    gameWindow = new GameWindow(gamePanel);
    gamePanel.setFocusable(true);
    gamePanel.requestFocus();

    startGameLoop();
}
```

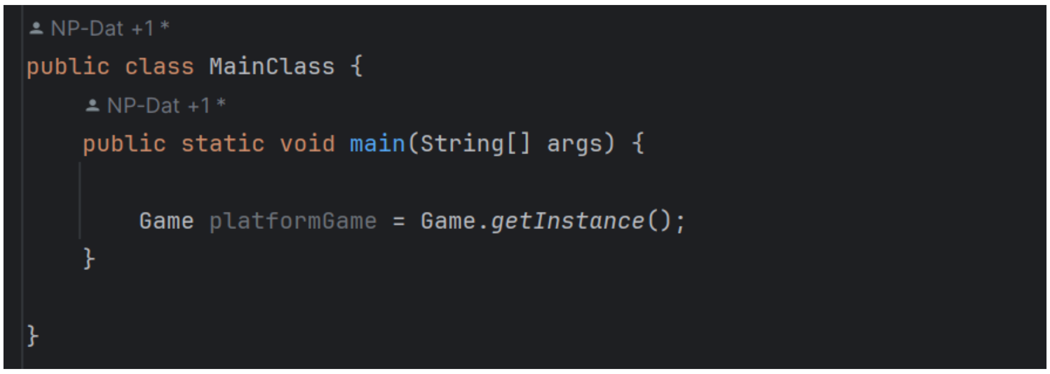
Figure 3.13



```
2 usages  thaortrinh
public static Game getInstance() {
    if (instance == null) {
        instance = new Game();
    }
    return instance;
}
```

Figure 3.14

- In MainClass class:

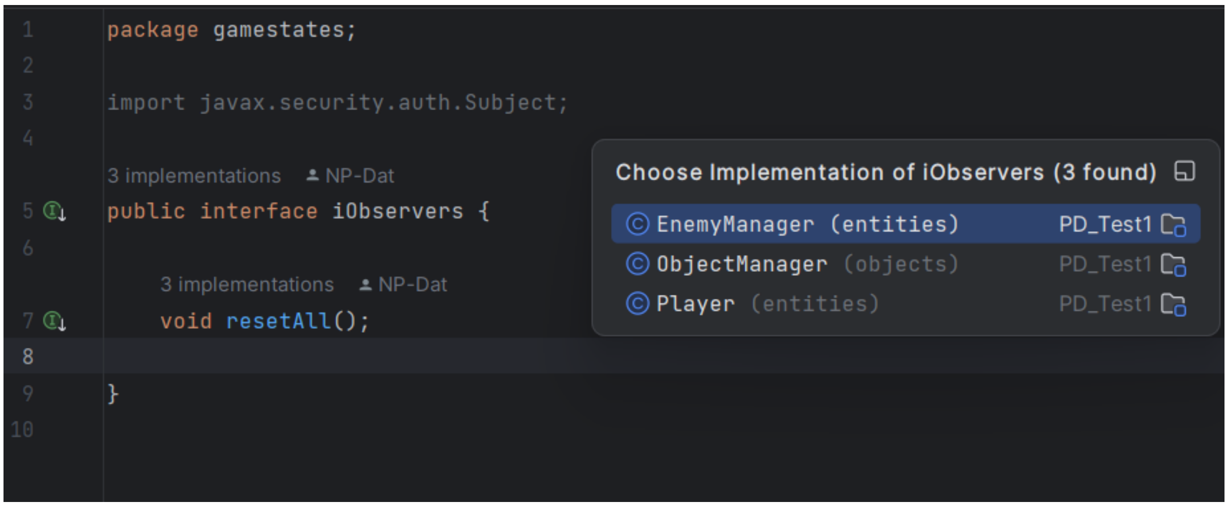


```
NP-Dat +1 *
public class MainClass {
    NP-Dat +1 *
    public static void main(String[] args) {

        Game platformGame = Game.getInstance();
    }

}
```

Figure 3.15

- **Observer**

- In "gamestates" package, we use observer pattern for Playing, Player, EnemyManager, ObjectManager class. The Playing class is the subject and when initialize other objects, it will also add Player, EnemyManager, ObjectManager objects into Arraylist<iObservers> observers. The subject will control all the observers in resetAll() function.
- IObserver interface:

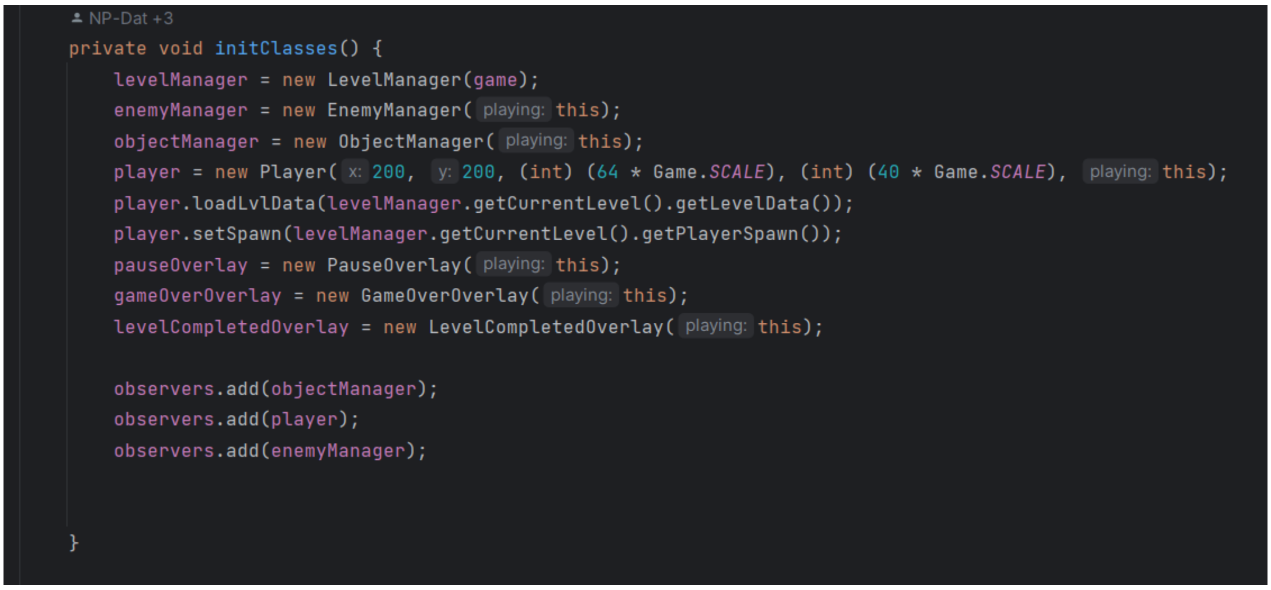


```java
package gamestates;

import javax.security.auth.Subject;

3 implementations  NP-Dat
public interface iObservers {

    3 implementations  NP-Dat
    void resetAll();

}
```

Choose Implementation of iObservers (3 found)
EnemyManager (entities) PD_Test1
ObjectManager (objects) PD_Test1
Player (entities) PD_Test1

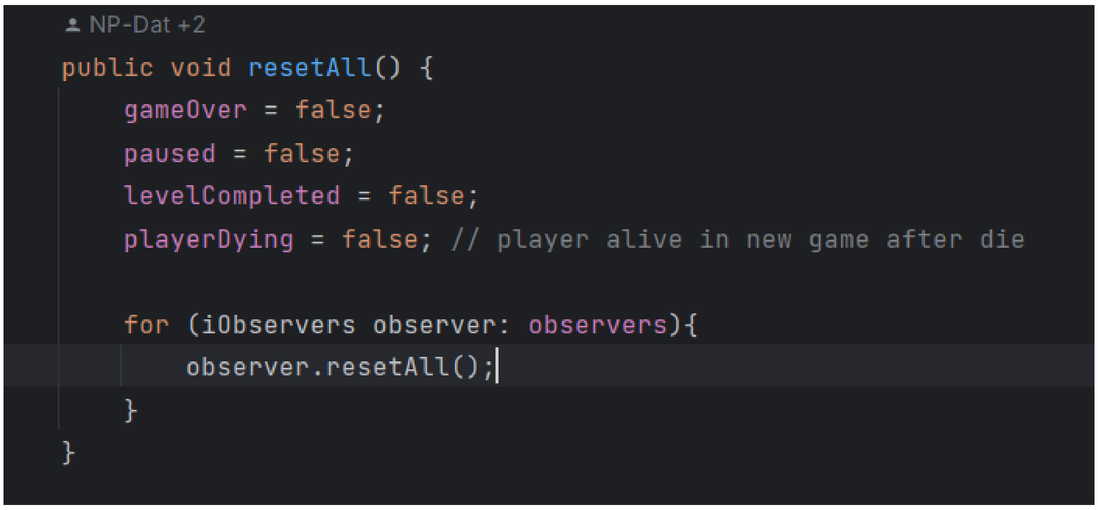
Figure 3.16: iObservers interface

- In Playing class:



```java
NP-Dat +3
private void initClasses() {
    levelManager = new LevelManager(game);
    enemyManager = new EnemyManager( playing: this);
    objectManager = new ObjectManager( playing: this);
    player = new Player( x: 200,  y: 200, (int) (64 * Game.SCALE), (int) (40 * Game.SCALE),  playing: this);
    player.loadLvlData(levelManager.getCurrentLevel().getLevelData());
    player.setSpawn(levelManager.getCurrentLevel().getPlayerSpawn());
    pauseOverlay = new PauseOverlay( playing: this);
    gameOverOverlay = new GameOverOverlay( playing: this);
    levelCompletedOverlay = new LevelCompletedOverlay( playing: this);

    observers.add(objectManager);
    observers.add(player);
    observers.add(enemyManager);


}
```

Figure 3.17: Adding observer object into ArrayList

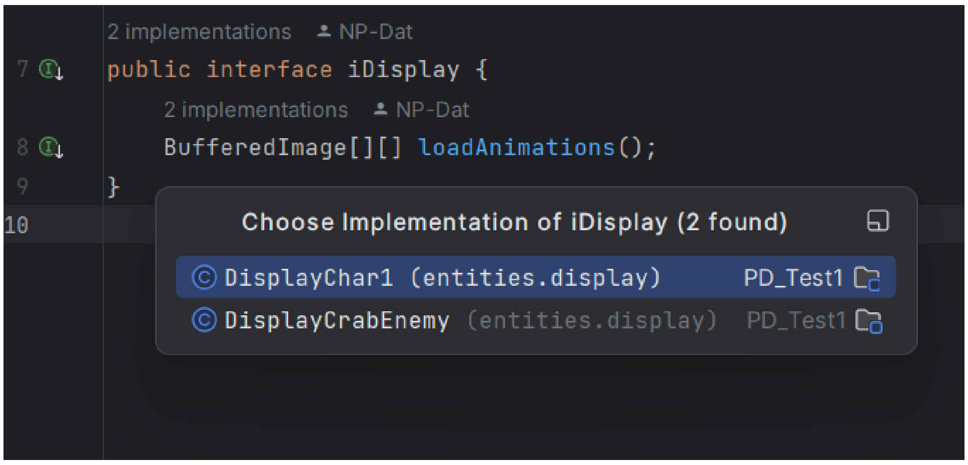


```java
NP-Dat +2
public void resetAll() {
    gameOver = false;
    paused = false;
    levelCompleted = false;
    playerDying = false; // player alive in new game after die

    for (iObservers observer: observers){
        observer.resetAll();
    }
}
```

Figure 3.18: resetAll() method notifying all observers of Playing class

- **Strategy**

- In entities package, we encapsulate the loadAnimations() method by using an interface called iDisplay. For each loadAnimations() method for each design of entities, there is a class which implements iDisplay and overwrites loadAnimations(). So that, it is easier to maintain when someone wants to switch between many designs.
- IDisplay interface:



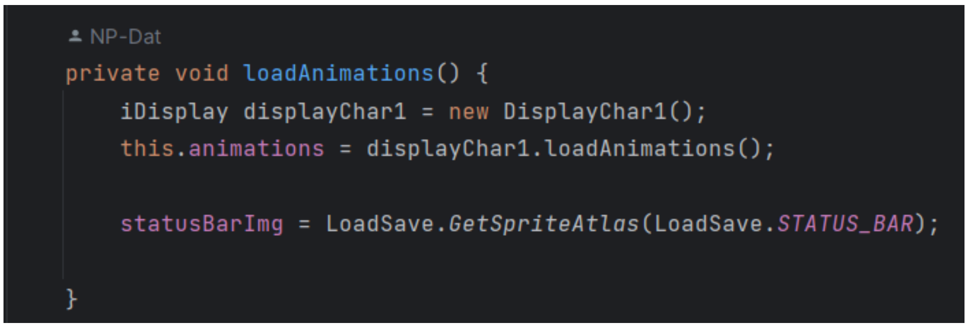
```
2 implementations  NP-Dat
public interface iDisplay {
    2 implementations  NP-Dat
    BufferedImage[][] loadAnimations();
}
```

Choose Implementation of iDisplay (2 found)

```
DisplayChar1 (entities.display)       PD_Test1
DisplayCrabEnemy (entities.display)   PD_Test1
```

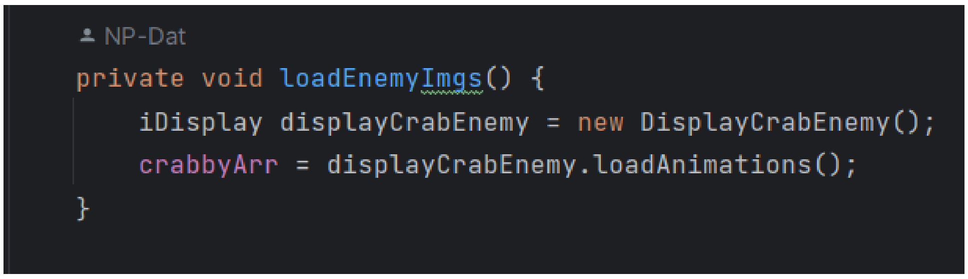
Figure 3.19: iDisplay interface

- Player and EnemyManager implementing loadAnimations:



```
NP-Dat
private void loadAnimations() {
    iDisplay displayChar1 = new DisplayChar1();
    this.animations = displayChar1.loadAnimations();

    statusBarImg = LoadSave.GetSpriteAtlas(LoadSave.STATUS_BAR);

}
```

Figure 3.20: loadAnimations() method in Player class



```
NP-Dat
private void loadEnemyImgs() {
    iDisplay displayCrabEnemy = new DisplayCrabEnemy();
    crabbyArr = displayCrabEnemy.loadAnimations();
}
```

Figure 3.21: loadEnemyImgs() method in EnemyManager class

IV. CLASS DIAGRAMS

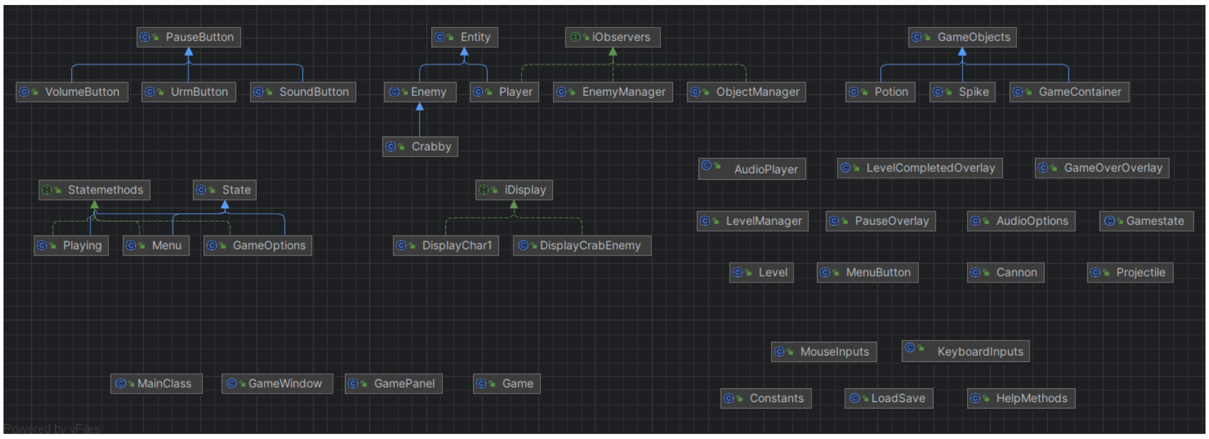


Figure 4.1: Class diagram

V. DEMONSTRATION

Let’s delve into the demonstration of the game, where we take you through the exciting playthrough of our game. In this hands-on experience, you will witness the special gameplay stats that make our game interesting. The game includes the following states.

Link to our gameplay demo: https://youtu.be/ZhAexeF6MU4

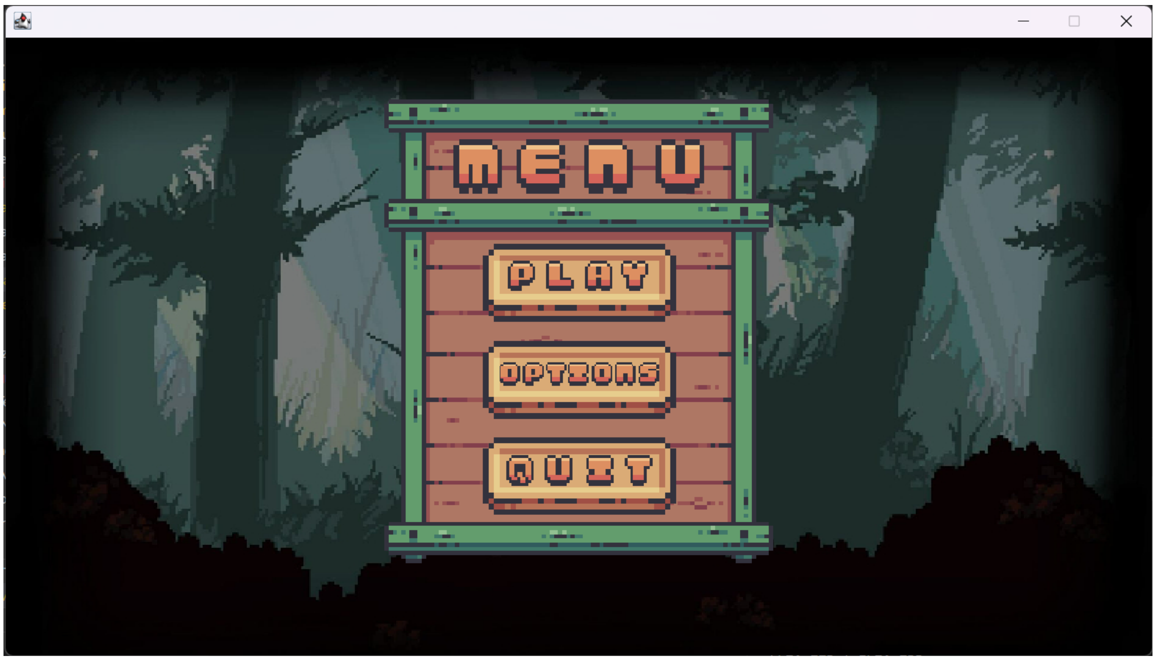


Figure 5.1: Menu

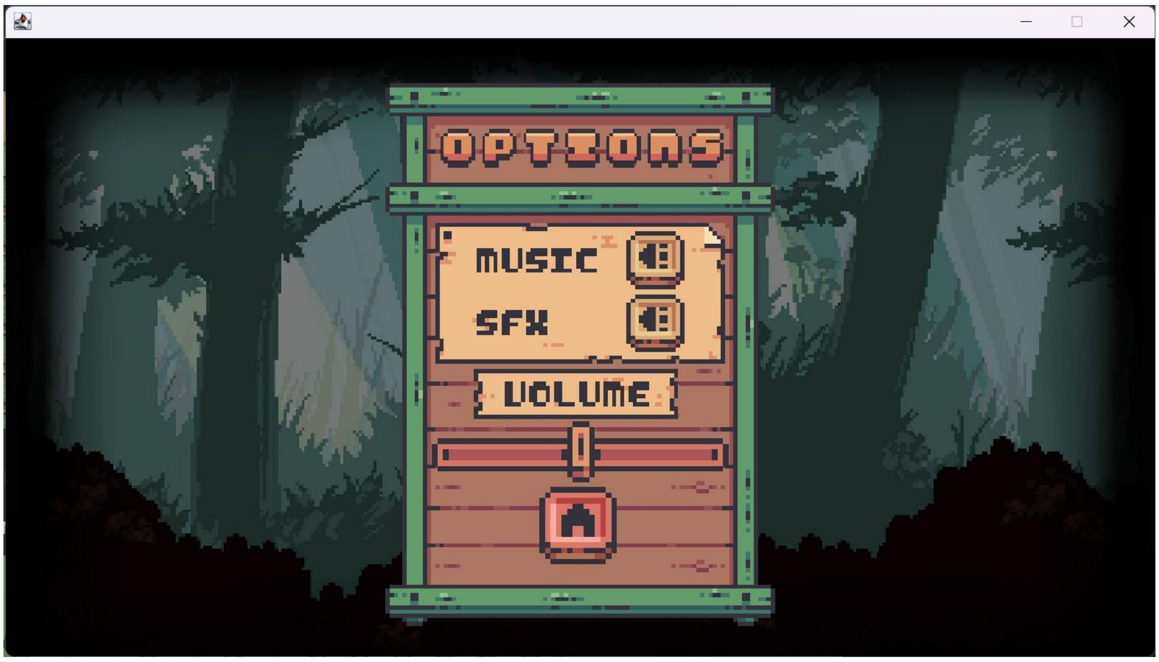


Figure 5.2: Game Options

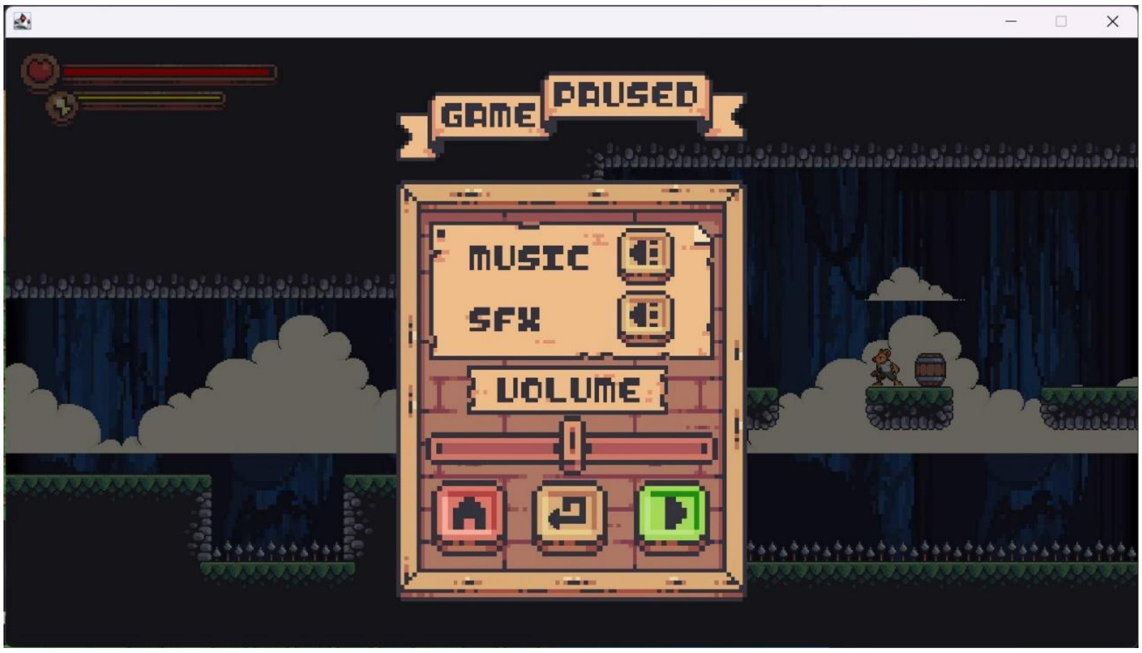


Figure 5.3: Game pause

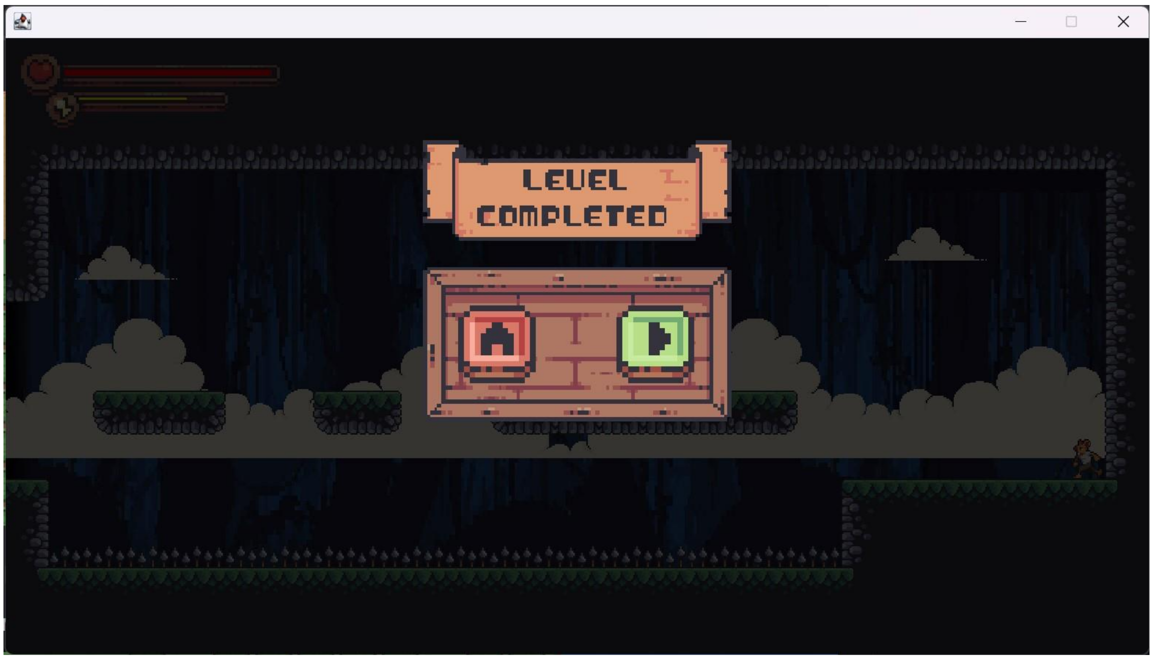


Figure 5.4: Level completed

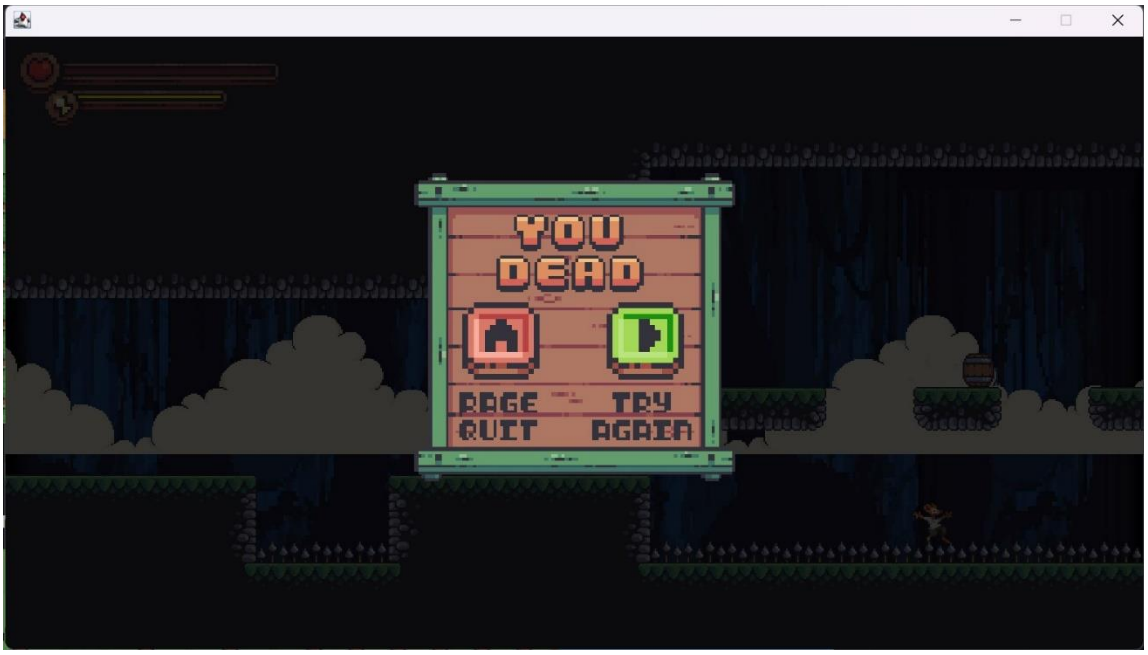


Figure 5.5: Game Over

VI. CONCLUSION

1. Summary

This project gave us a chance to learn more about OOP, how to implement it into practice like building a game, as well as approach the way people control and manage code for a big project. Our team has followed the tutorial and try to implement it with many different assets such as player sprite and game background, understand how to draw pixels map with Piskel,...
We also learn about project management with GitHub, and how to store, manage, track and control changes to our code.
Above all, the project has shown us the way the concept of object-oriented programming has been concretely applied to build a game, as well as following SOLID principles and design pattern such as Singleton and Observer.

2. Limitation

However, there are also some limitations that our team has committed: limited programming experience, unfamiliarity with game development concept and narrow knowledge in design patterns.

3. Evaluation

Nevertheless, with the adequate understanding about building and implementing the game, our team design the game that could be extends to have more levels with more maps, a level can be added with more enemies and traps.
The character can also be replaced with your own custom, and you need to combine the sprite images to a 2-dimension array.
The skills of enemies can be upgraded with more strength, which can increase the difficulty of game levels.

REFERENCES

1. **Tutorial**

https://www.youtube.com/@KaarinGaming

2. **GitHub repository examples**

https://github.com/sang-bui20501/ProjectOOP-daRIEUnaruto

https://github.com/tientrinh2003/CodingGame-OFFICIAL?fbclid=IwAR35Z2_2Y2qufuxzgMrI99-WVXVkZBWOAQAkSvHhKfg4T_0i1VOr-CKdXks

3. **Design patterns**

https://www.javacodegeeks.com/2015/09/singleton-design-pattern.html

https://www.javacodegeeks.com/2015/09/observer-design-pattern.html

https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm