FINAL REPORT

Course: Object Oriented Programming

INSTRUCTOR:

Assoc. Prof. Tran Thanh Tung and Nguyen Trung Nghia

TEAM MEMBER:
Nguyen Minh Khoi – ITCSIU22210
Vuong Quan Sieu – ITCSIU22270
Truong Huy Hoang – ITCSIU22228
Huynh Trinh Phuc Thinh – ITCSIU22230

Table of Contents

Chap 1: Introduction

1.1 Introduction to the Game

In today's technology-driven society, gaming has become increasingly popular and well-developed. Therefore, we implement a project in order to create a game that not only entertains but also practices and applies knowledge.

When it comes to tower defense games, Bloons Tower Defense (BTD) is undoubtedly one of the most iconic names. We chose this topic because of its enduring appeal, from its simple yet captivating gameplay to the endless creativity it inspires.

First released in 2007 by Ninja Kiwi, Bloons Tower Defense has undergone numerous upgrades, with each new version bringing fresh challenges and excitement to players. The game is set in a world where monkeys must defend against invading balloons (bloons), offering a unique blend of strategy and entertainment. Its simple storyline, combined with clever design, makes every level feel like a distinct strategic challenge.

Through this project, we aim not only to recreate the engaging gameplay of BTD but also to improve essential skills, such as programming skills: Utilizing programming languages and game development tools to build the project.

UI/UX design skills: Creating an appealing and immersive player experience.

Strategic and logical thinking: Designing gameplay features that are both captivating and balanced.

1.2 About the Project

The "Bloons Tower Defense" project is a strategy-based game in Java, wherein towers of different types have to be placed along a way to defend against waves of enemies from reaching the base. Core idea: Design a system such that players will have to think strategically, balancing tower placement and upgrade choices with enemy management.

The aim of the project is to implement an interactive, fully functional tower defense game. A player can defend the base by placing and upgrading towers on a playing field. The purpose is to prevent waves of enemies from reaching the base by building more powerful towers and constructing effective defense plans.

Our project is an improved version inspired by the original Bloons Tower Defense. The goal is to replicate the charm of the original game while adding creative elements to enhance the player experience.

1.3 Our Bloon Tower Defense

Building upon the original Bloons Tower Defense, we aim to:

Retain familiar elements such as the monkey characters, balloons, and tower defense gameplay.

Introduce new features, including:

- Unique weapons and skills for each monkey.
- Special balloons with their own challenges.

This chapter provides an overview of our project, covering the reasons behind our choice, the history of Bloons Tower Defense, and the standout features we plan to develop in our version.

1.4 References:

Image from: https://www.pixilart.com/draw/shuriken-4ddba5bc1f994be
https://bloons.fandom.com/wiki/Category:BTD4_Assets
https://www.spriters-resource.com/pc_computer/bloonstowerdefense6/
https://github.com/harrisbchong/Bloons-Tower-Defense-Game/tree/main/Final%20Project/res
Tutorial Java Development Game:
https://www.youtube.com/watch?v=kclnyiXmY7Q&list=PL4rzdwizLaxb0-TajNIp5DOoT_PAxhx0T

Developer team:

Member name **- Github username**	**UID**	**Contribution**
Nguyen Minh Khoi - enemkayy	**ITCSIU22210**	**100%**
Vuong Quan Sieu - VSieu	**ITCSIU22270**	**100%**
Truong Huy Hoang - hoanghaiphong15	**ITCSIU22228**	**100%**
Huynh Trinh Phuc Thinh -tuilathinh1	**ITCSIU22230**	**100%**

Chap 2: System Design

2.1 List of class and responsibility

Package	Class Name	Responsibility
Enemies	Enemy	This is an abstract class that serves as the base for all enemy types.
	Rbl	This is specific enemy (redbloon) implement extend the enemy class.
	Bbl	This is specific enemy (bluebloon) implement extend the enemy class.
	Pbl	This is specific enemy (purplebloon) implement extend the enemy class.
	Spl	This is specific enemy (specialbloon) implement extend the enemy class.
events	Wave	This class plays a role in manage enemy spawn
helpz	Constant	This is class that provide static final constants for directions, enemy types, tiles, projectiles, tower and monkey.
	ImgFix	This class to rotate the images, layers and combining them.
	LoadSave	This class to loads images and assets from resource folder. Create and read level data files
	Utilz	Converts between 1D and 2D arrays. Computes directional arrays for roads in game levels. Calculates distances and validates tile properties.

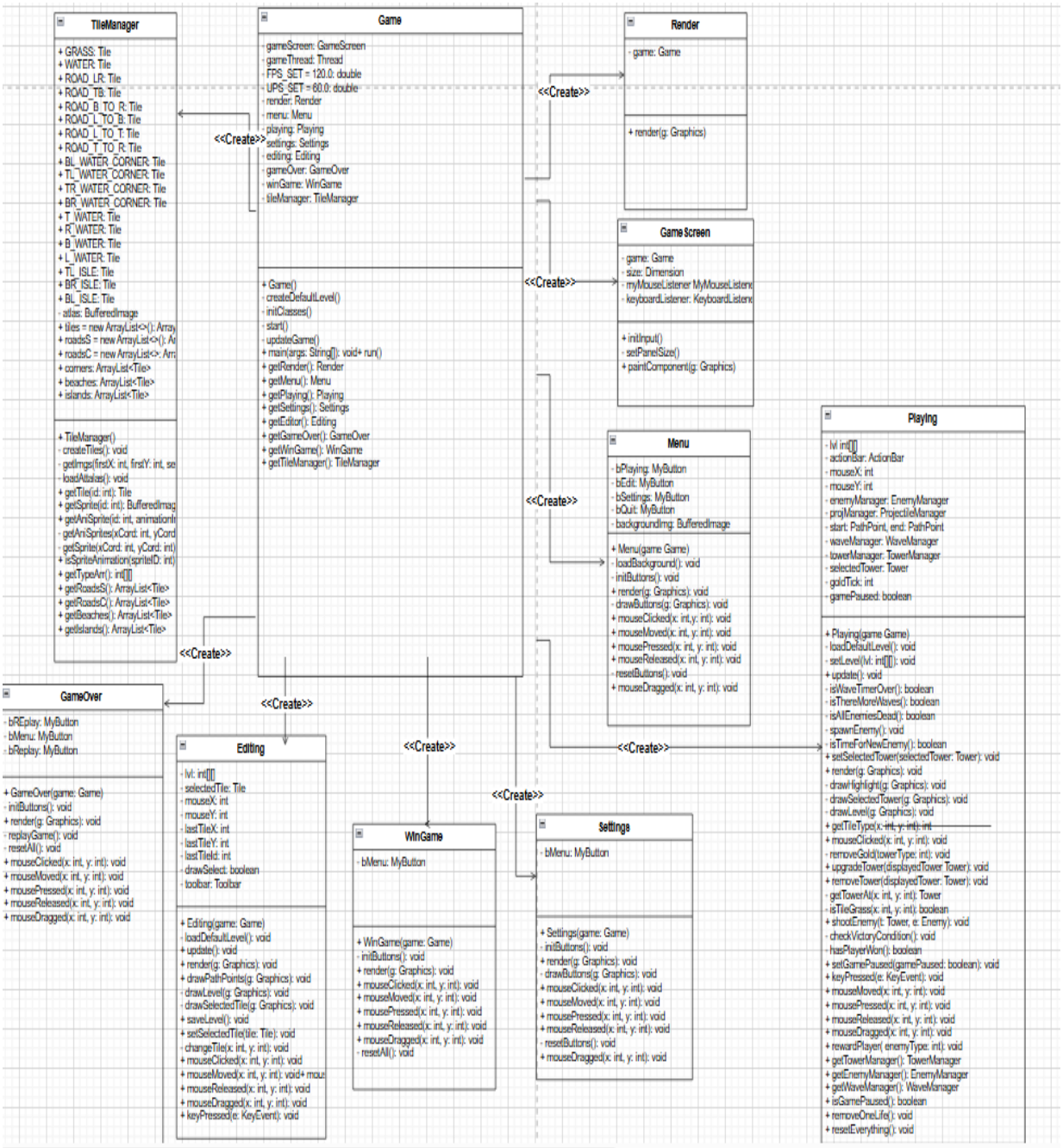
inputs	KeyboardListener	Implements the KeyListener interface to handle keyboard events.
	MyMouseListener	Implements both MouseListener and MouseMotionListener interfaces to handle mouse events.
main	Game	This class acts as the core of the game application. Handles the game loop for rendering (FPS) and updating (UPS). Creates the main game window and sets up default settings.
	GameScreen	This class handles the display panel for redering the game.
	GameStates	This class represents the different states of the game. Maintains the current game state through the gameState static field.
	Render	This class manages rendering logic for the game.
managers	EnemyManager	This class spawns new enemies base on game logic. Draws, updates enemies positions and states on screen.
	ProjectileManager	Spawns and tracks projectiles, including their movement and collision detection with enemies.
	TileManager	This class handles the visual and logical representation of game tiles.
	TowerManager	This oversees tower placement, upgrades, and attacks.
	WaveManager	This class organizes and tracks enemy waves and spawning logic.

objects	PathPoint	This class represents a point on a path in the game, typically used for enemy movement or pathfinding.
	Projectile	This class Updates position and moves. Tracks the position, speed, damage.
	Tile	Stores tile’s ID, type and associated images
	Tower	Tracks the tower's position, type, damage, range, cooldown, and tier (upgrade level).
scenes	Editing	Allows players to modify the game map by selecting and placing tiles. Tracks mouse interactions for editing the level layout.
	GameOver	Displays the "Game Over" message and options to replay or return to the menu.
	GameScene	Serves as a base class for all scenes in the game.
	Menu	Manages the game's main menu.
	Playing	Tracks and updates enemies, towers, projectiles, waves, and the action bar.
	SceneMethods	Interface that establishes a consistent interface for rendering and handling mouse interactions.
	Settings	Provides an interface to adjust game settings.
	WinGame	Represent end-of-game states.
	ActionBar	Displays and manages game-related stats and tower interactions.

ui	Bar	Provides a foundation for UI components.
	MyButton	Represents and handles buttons in the UI.
	Toolbar	Supports level editing with tools for tile selection and placement.

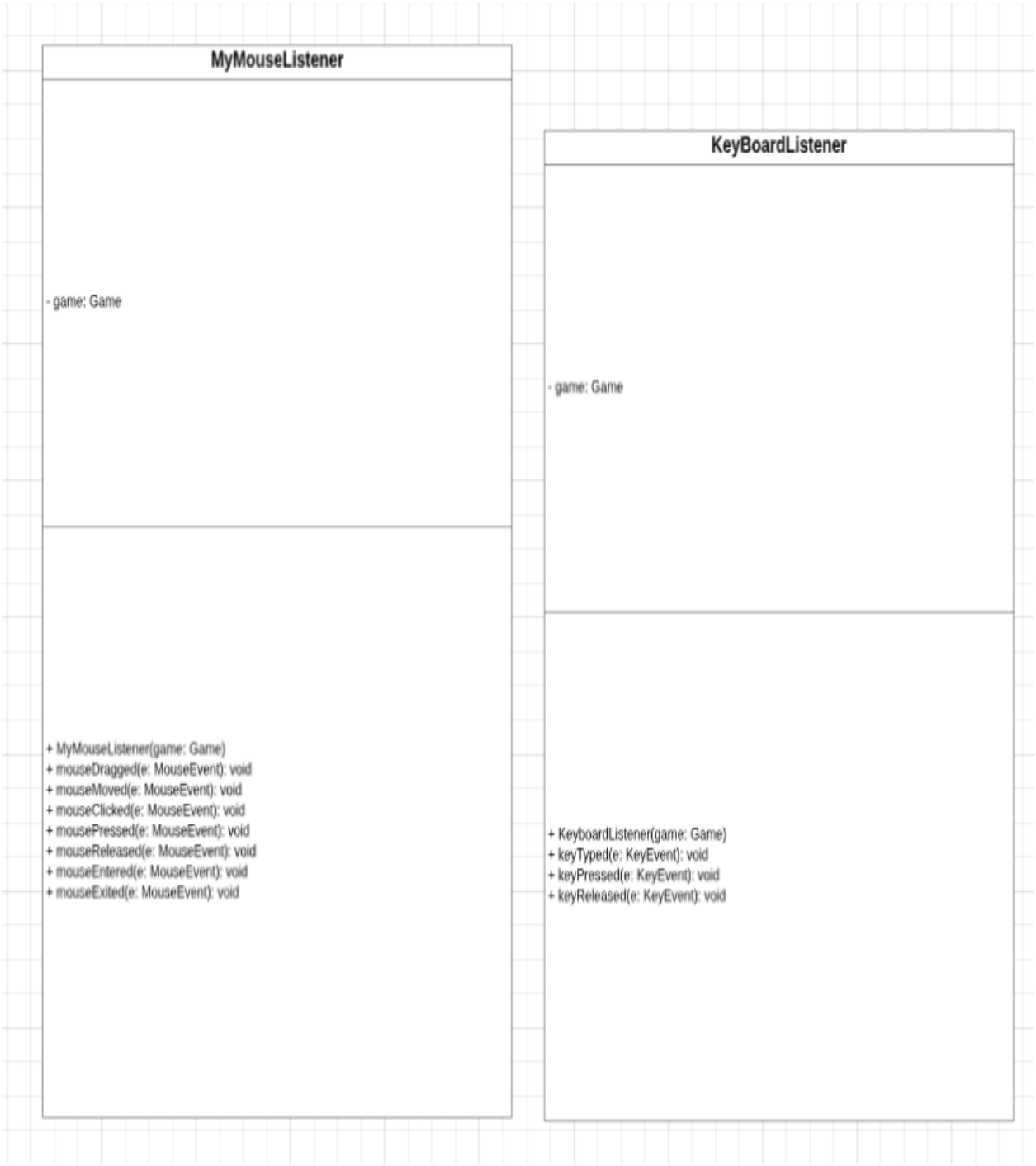
2.2 UML class diagram

Main Class diagram



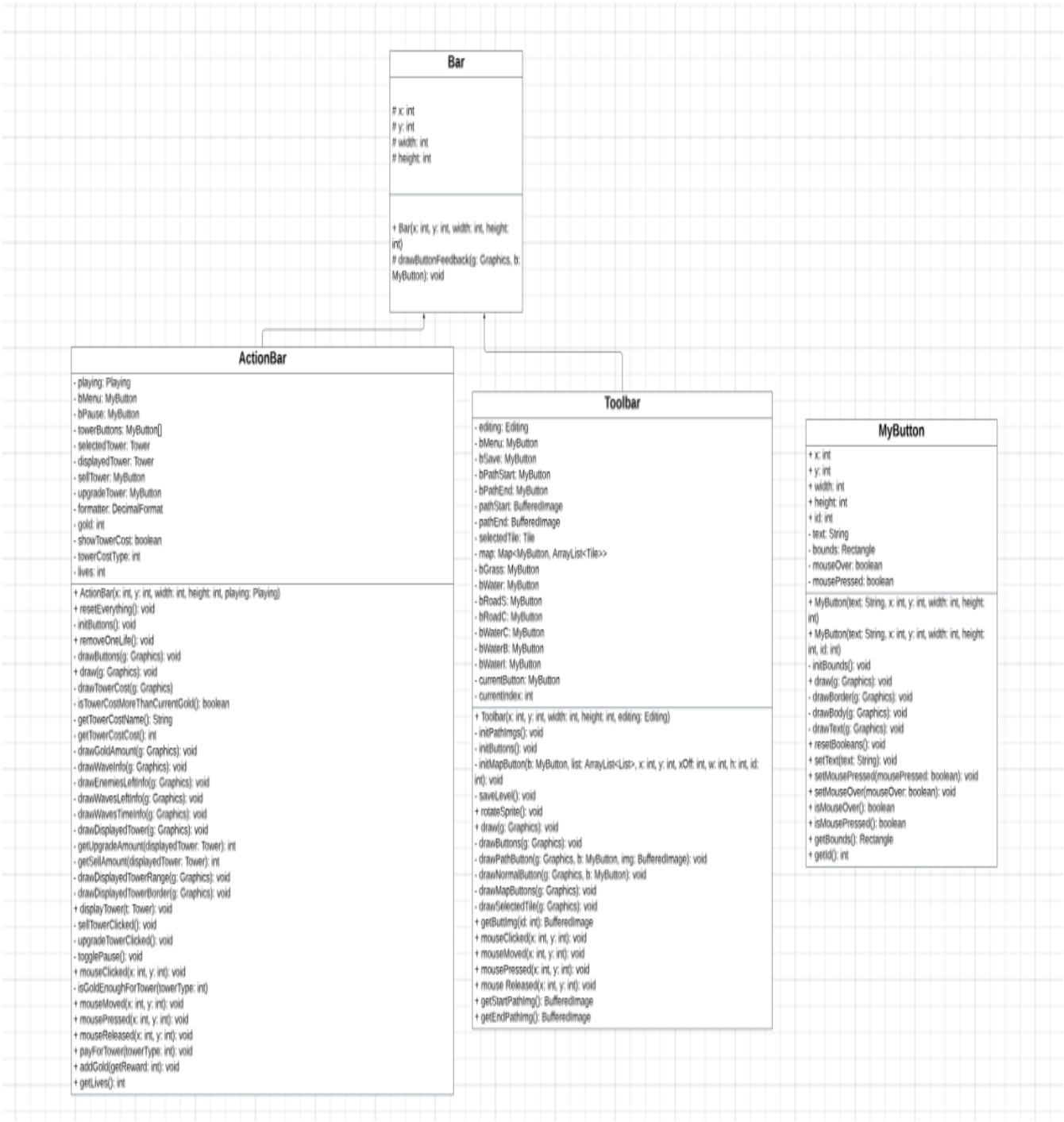
The Game class orchestrates the game by managing initialization, scenes, rendering, updates, and transitions between states like PLAYING and MENU, while ensuring smooth execution via a game loop.

Input diagram



The KeyboardListener and MyMouseListener classes handle user inputs, such as key presses, mouse movements, and clicks. They delegate input actions to the appropriate game states (MENU, PLAYING, EDIT, etc.), ensuring each state responds correctly based on the player's interactions.

User Interface diagram

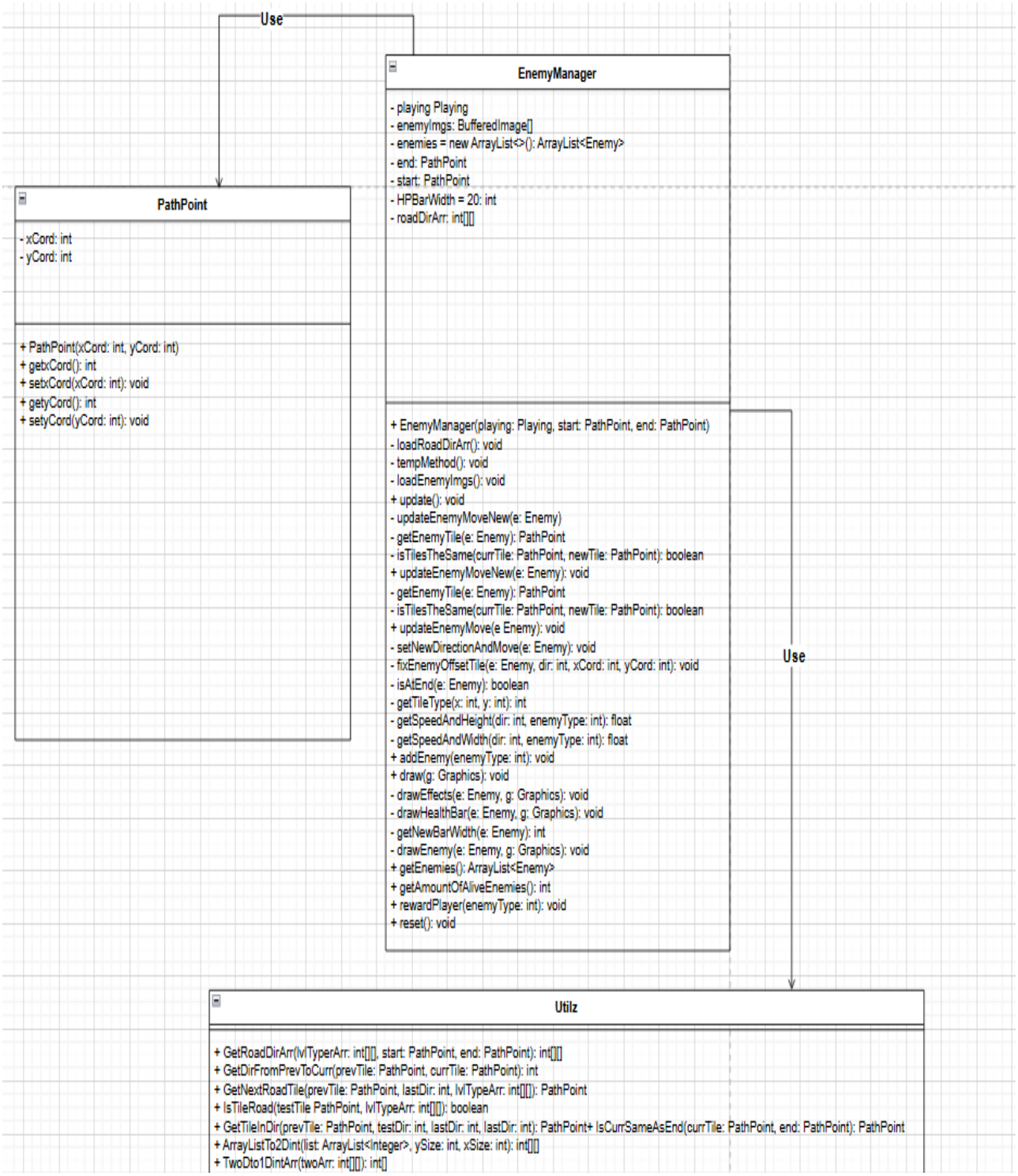


The user interface (UI) classes in the code manage game information display (lives, gold, wave progress), tower details (cost, upgrades, and selling), and interactive elements like buttons for game controls (pause, menu, actions). They handle player interactions, ensuring smooth communication between the player and the game logic.

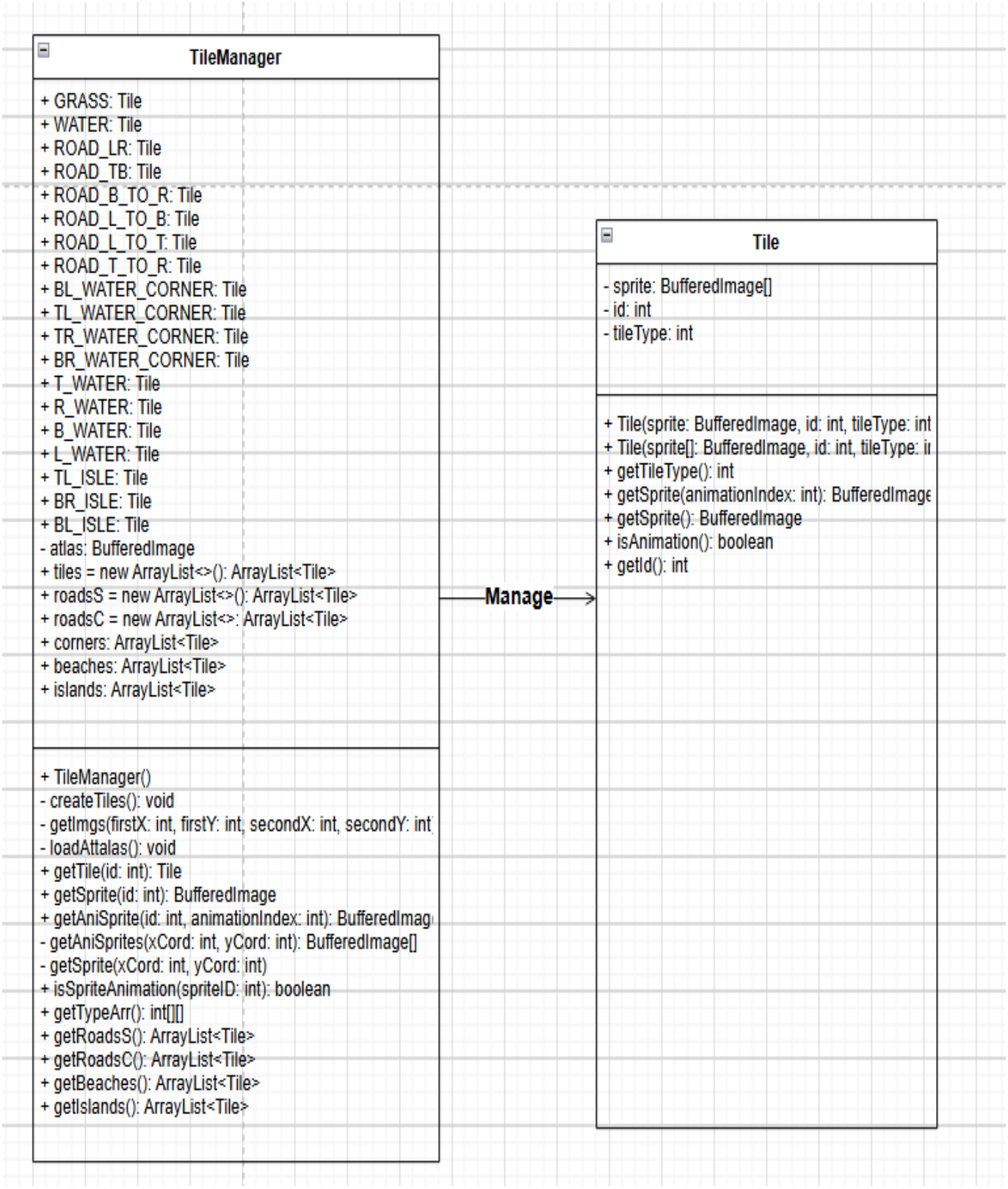
Scenes diagram

The scenes package manages game stages like Editing, Playing, Menu, and GameOver. Each class handles specific functions such as rendering, inputs, and logic. Editing allows level creation, Playing manages gameplay, and Menu and GameOver handle navigation and end-game actions. Shared logic comes from GameScene.

Pathfinding diagram

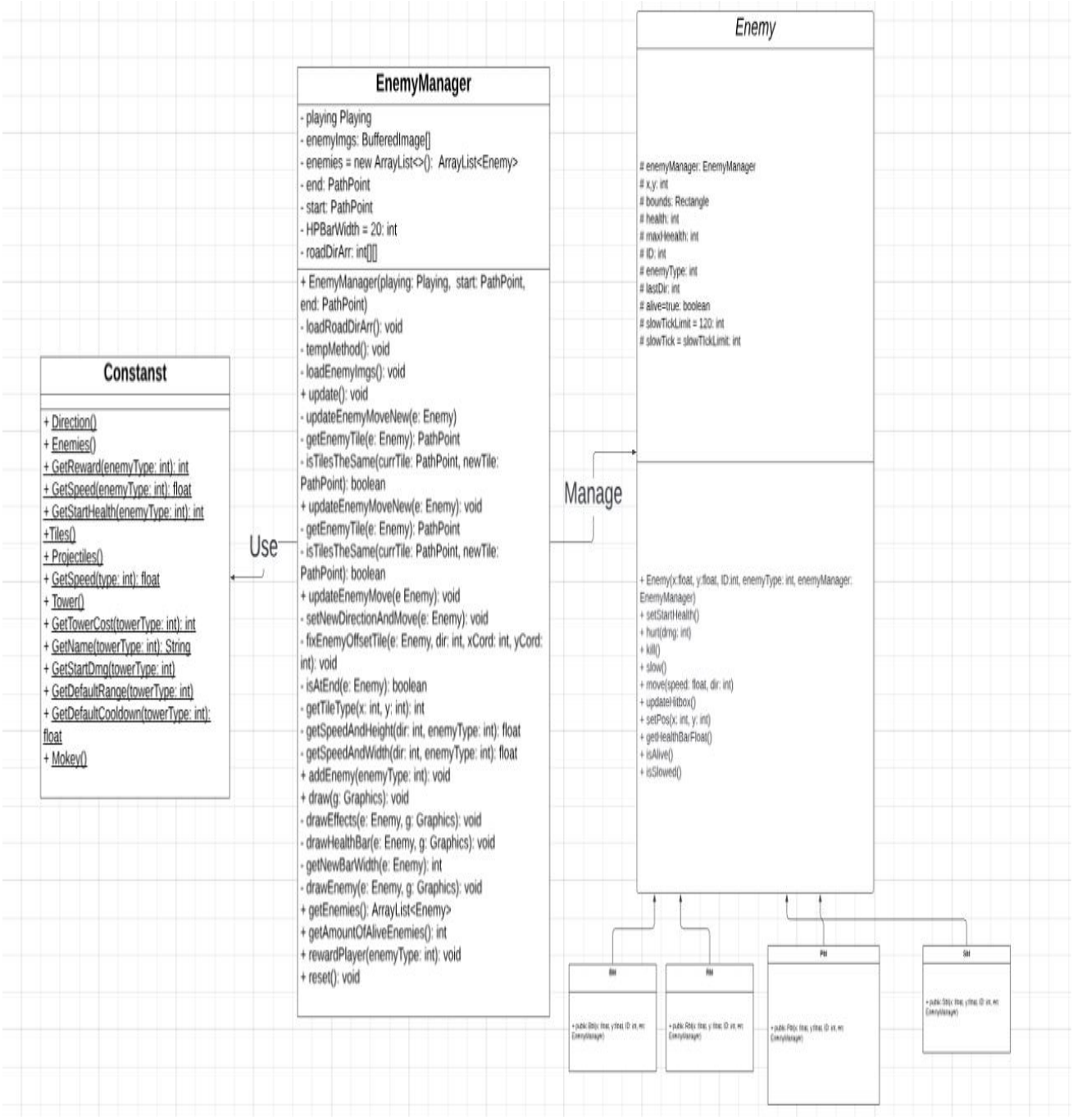


The pathfinding classes manage enemy navigation on game paths. ***PathPoint*** *holds tile coordinates.* ***Utilz*** *calculates the road direction, finds the next valid tile, and converts arrays.* ***EnemyManager*** *moves enemies based on this data, updates positions, and manages game interactions like life reduction. Together, they enable enemy path movement.*
Tile diagram



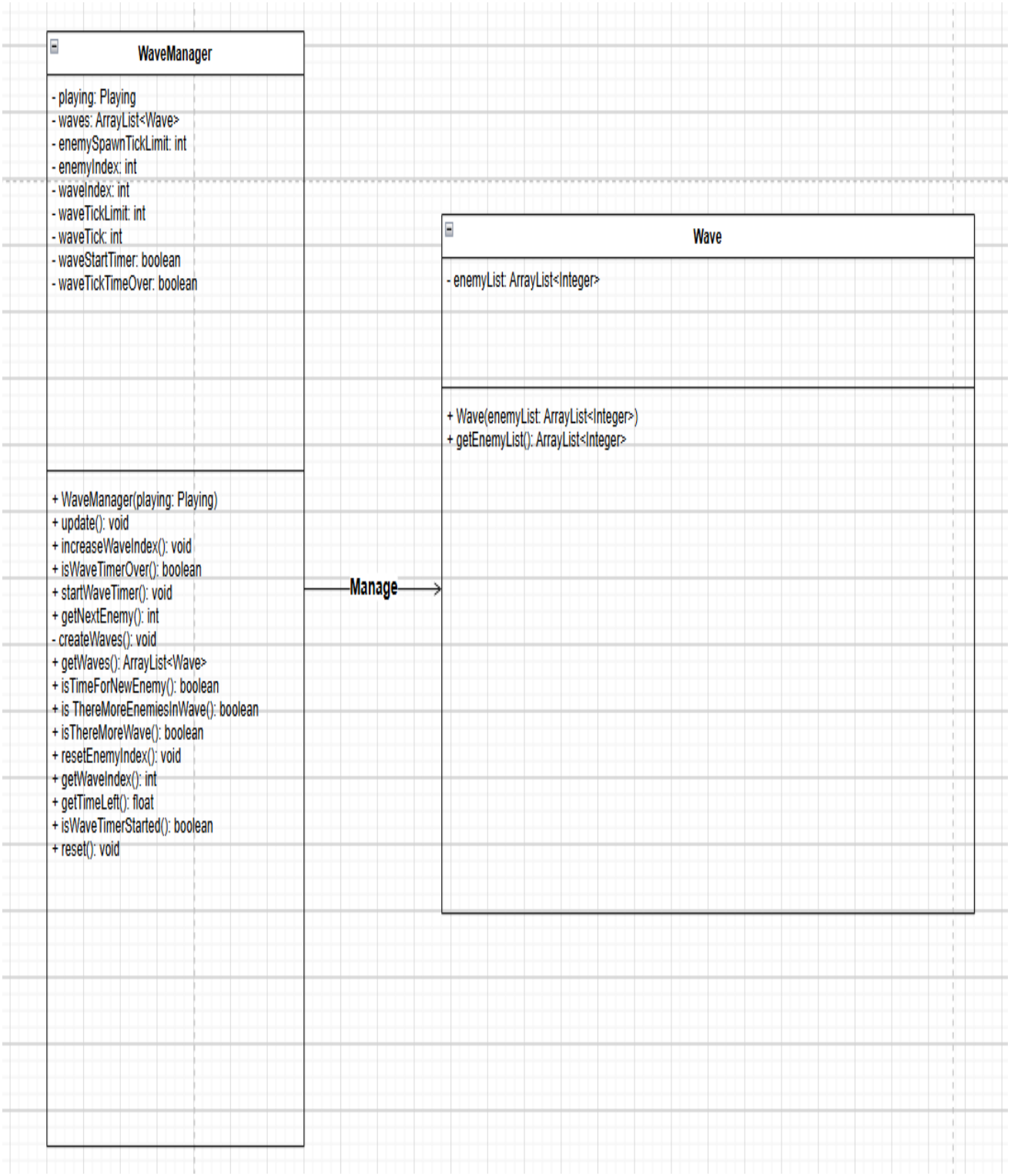
TileManager loads, organizes, and manages tiles, giving each a type and ID for easy use in the game.

Enemy diagram

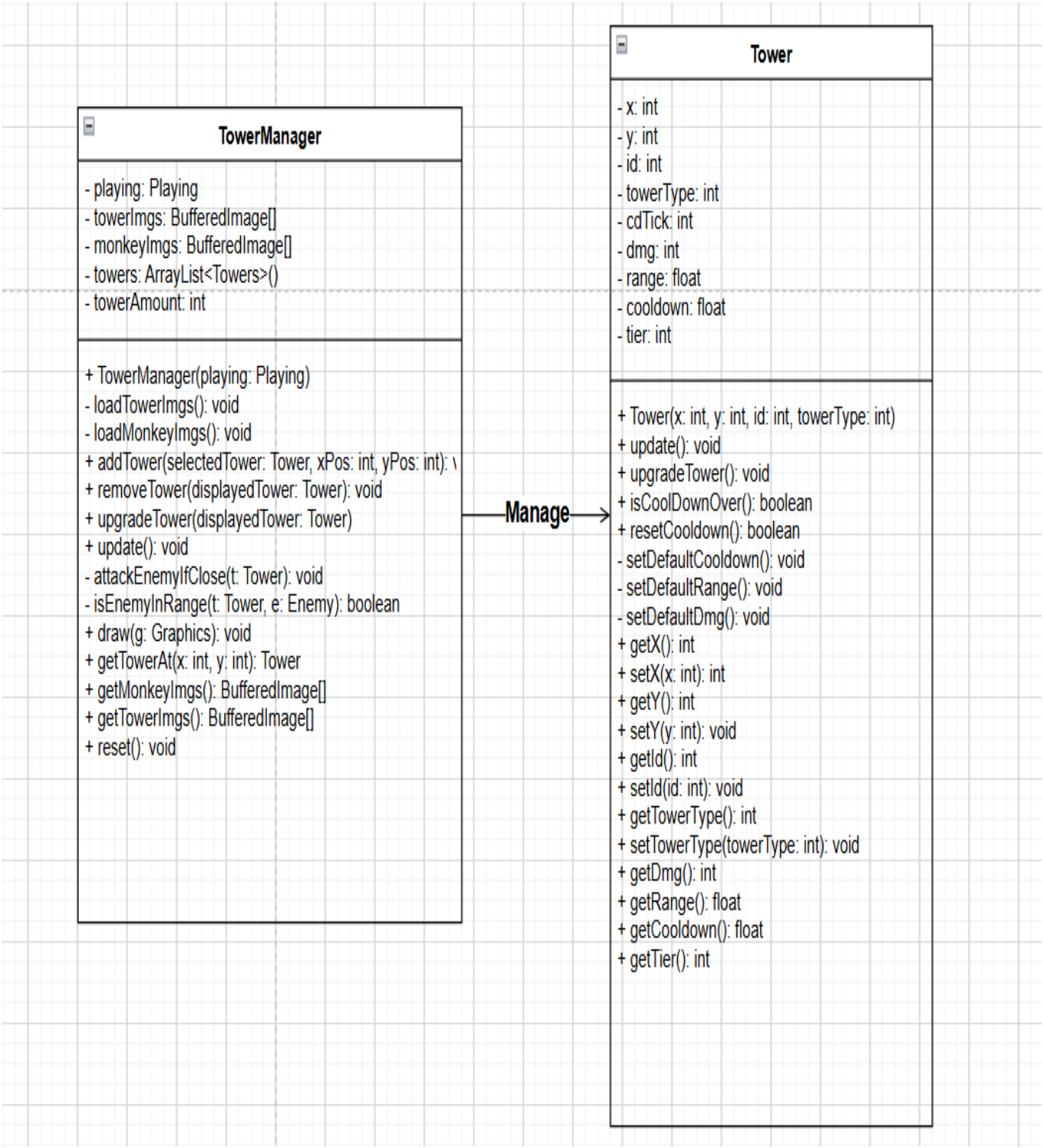


*The **EnemyManager** handles enemy management: loading images, controlling movement, applying effects, drawing enemies, and updating their state. It manages interactions with the game, such as rewarding the player or deducting lives when enemies reach the endpoint. The **Enemy** class represents individual enemies, providing base functionality like movement, health management, and collision detection. Subclasses like **Rbl**, **Bbl**, **Pbl**, and **Sbl** define specific enemy types, inheriting base behavior from Enemy.*

Wave diagram

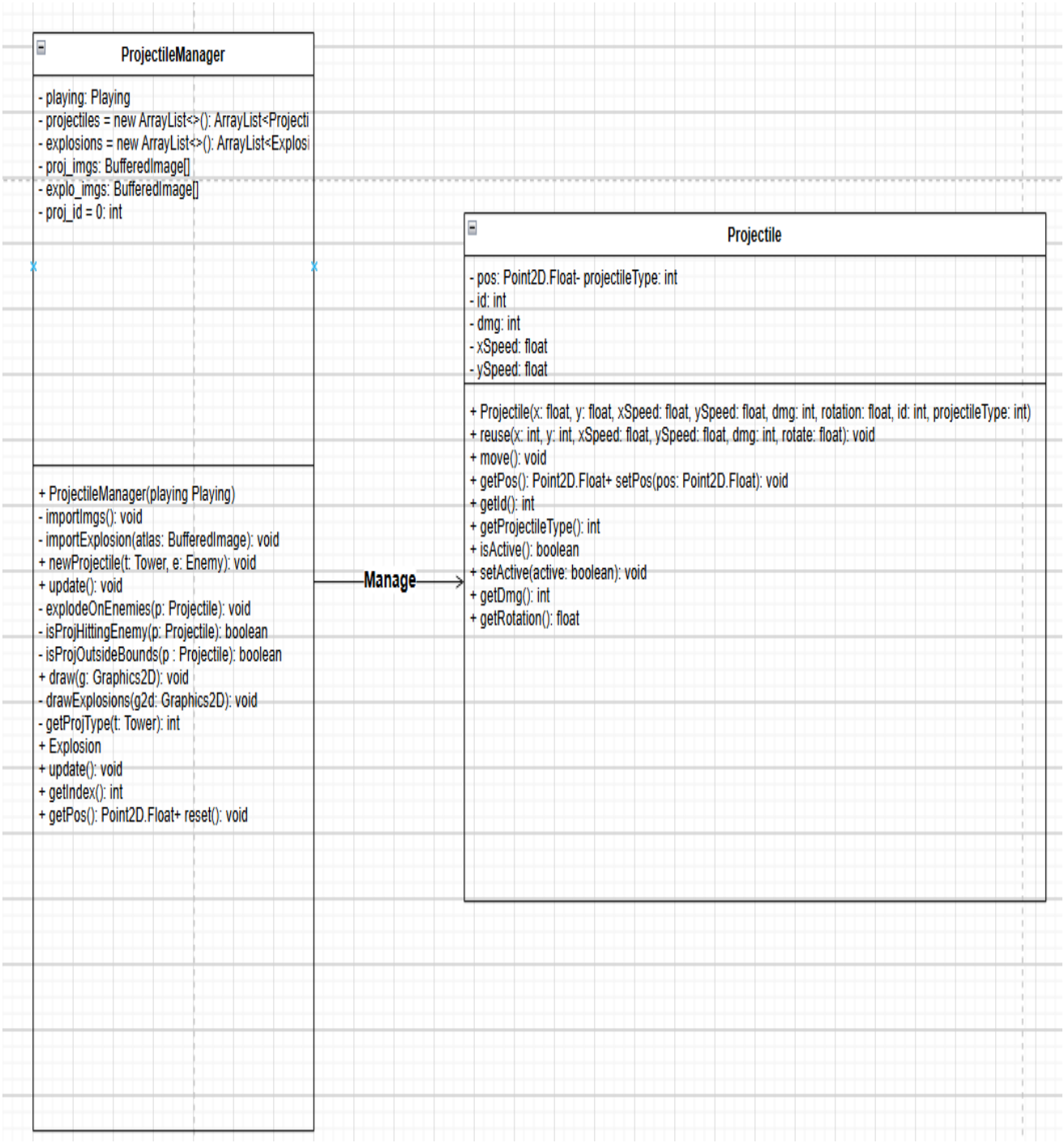


*The **WaveManager** controls wave progression, spawning enemies based on a timer, managing wave intervals, and resetting wave data. It tracks the current wave, enemy spawn timing, and handles transitions between waves. The **Wave** class represents a single wave, storing a list of enemy types for that wave.*
Tower diagram



*The **TowerManager** handles the creation, management, upgrading, and removal of towers, as well as attacking enemies within range. It updates towers, checks attack conditions, and controls visuals. The **Tower** class defines individual towers with attributes like position, type, damage, range, cooldown, and upgrade mechanics. Together, they implement tower functionality and interactions in the game.*

Projectile diagram



*The **ProjectileManager** oversees the creation, updating, and rendering of projectiles and explosions in the game. It calculates projectile trajectories, handles collisions with enemies, and manages explosive effects. The **Projectile** class represents individual projectiles, tracking their position, speed, damage, and state (active/inactive). Together, these classes manage the mechanics of projectiles and their interactions with enemies.*

Chap 3: Implementation Code

3.1 Game Initialization

The game initializes by setting up the window, play area, and essential components.

```
public Game() {

    initClasses();
    createDefaultLevel();

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLocationRelativeTo(null);
    setResizable(false);
    setTitle("Bloon Tower Defense Remake");
    add(gameScreen);
    pack();
    setVisible(true);

}
 public static void main(String[] args) {

     Game game = new Game();
     game.gameScreen.initInputs();
     game.start();

 }
```

3.2 Game Loop

The game loop updates and renders the game at 120 FPS

```java
@Override
public void run() {

    double timePerFrame = 1000000000.0 / FPS_SET;
    double timePerUpdate = 1000000000.0 / UPS_SET;
    long lastFrame = System.nanoTime();
    long lastUpdate = System.nanoTime();
    long lastTimeCheck = System.currentTimeMillis();
    int frames = 0;
    int updates = 0;
    long now;

    while (true) {
        now = System.nanoTime();

        // Render
        if (now - lastFrame >= timePerFrame) {
            repaint();
            lastFrame = now;
            frames++;
        }

        // Update
        if (now - lastUpdate >= timePerUpdate) {
            updateGame();
            lastUpdate = now;
            updates++;
        }

        if (System.currentTimeMillis() - lastTimeCheck >= 1000) {
            System.out.println("FPS: " + frames + " | UPS: " + updates);
            frames = 0;
            updates = 0;
            lastTimeCheck = System.currentTimeMillis();
        }

    }

}
```

3.3 Game State Management

Manages states like running, paused, and game-over.

```java
private void togglePause() {
    playing.setGamePaused(!playing.isGamePaused());

    if (playing.isGamePaused())
        bPause.setText("Unpause");
    else
        bPause.setText("Pause");

}

public void removeOneLife() {
    lives--;
    if (lives <= 0)
        SetGameState(GAME_OVER);
}
```

3.4 Tower Control

Handles player input for pausing the game, placing monkey towers, upgrading tower tiers, and selling towers

when they are no longer needed.

```
public void mouseClicked(int x, int y) {
    if (bMenu.getBounds().contains(x, y))
        SetGameState(MENU);
    else if (bPause.getBounds().contains(x, y))
        togglePause();
    else {

        if (displayedTower != null) {
            if (sellTower.getBounds().contains(x, y)) {
                sellTowerClicked();

                return;
            } else if (upgradeTower.getBounds().contains(x, y) && displayedTower.getTier() < 3
                    && gold >= getUpgradeAmount(displayedTower)) {
                upgradeTowerClicked();
                return;
            }
        }

        for (MyButton b : towerButtons) {
            if (b.getBounds().contains(x, y)) {
                if (!isGoldEnoughForTower(b.getId()))
                    return;

                selectedTower = new Tower(0, 0, -1, b.getId());
                playing.setSelectedTower(selectedTower);
                return;
            }
        }
    }

}

public void mouseMoved(int x, int y) {
    bMenu.setMouseOver(false);
    bPause.setMouseOver(false);
    showTowerCost = false;
    sellTower.setMouseOver(false);
    upgradeTower.setMouseOver(false);

    for (MyButton b : towerButtons)
        b.setMouseOver(false);

    if (bMenu.getBounds().contains(x, y))
        bMenu.setMouseOver(true);
    else if (bPause.getBounds().contains(x, y))
        bPause.setMouseOver(true);
    else {

        if (displayedTower != null) {
            if (sellTower.getBounds().contains(x, y)) {
                sellTower.setMouseOver(true);
                return;
            } else if (upgradeTower.getBounds().contains(x, y) && displayedTower.getTier() < 3) {
                upgradeTower.setMouseOver(true);
                return;
            }
        }

        for (MyButton b : towerButtons)
            if (b.getBounds().contains(x, y)) {
                b.setMouseOver(true);
                showTowerCost = true;
                towerCostType = b.getId();
                return;
            }
    }
}
```

```
public void mousePressed(int x, int y) {
    if (bMenu.getBounds().contains(x, y))
        bMenu.setMousePressed(true);
    else if (bPause.getBounds().contains(x, y))
        bPause.setMouseOver(true);
    else {

        if (displayedTower != null) {
            if (sellTower.getBounds().contains(x, y)) {
                sellTower.setMousePressed(true);
                return;
            } else if (upgradeTower.getBounds().contains(x, y) && displayedTower.getTier() < 3) {
                upgradeTower.setMousePressed(true);
                return;
            }
        }

        for (MyButton b : towerButtons)
            if (b.getBounds().contains(x, y)) {
                b.setMousePressed(true);
                return;
            }
    }

}

public void mouseReleased(int x, int y) {
    bMenu.resetBooleans();
    bPause.resetBooleans();
    for (MyButton b : towerButtons)
        b.resetBooleans();
    sellTower.resetBooleans();
    upgradeTower.resetBooleans();

}
```

3.5 Special features

3.5.1 Pathfinding

The GetRoadDirArr method calculates movement directions for each road tile in a 2D grid from a start to an endpoint. The result is a directional map guiding the path.

```
public static int[][] GetRoadDirArr(int[][] lvlTypeArr, PathPoint start, PathPoint end) {
    int[][] roadDirArr = new int[lvlTypeArr.length][lvlTypeArr[0].length];

    PathPoint currTile = start;
    int lastDir = -1;

    while (!IsCurrSameAsEnd(currTile, end)) {
        PathPoint prevTile = currTile;
        currTile = GetNextRoadTile(prevTile, lastDir, lvlTypeArr);
        lastDir = GetDirFromPrevToCurr(prevTile, currTile);
        roadDirArr[prevTile.getyCord()][prevTile.getxCord()] = lastDir;
    }
    roadDirArr[end.getyCord()][end.getxCord()] = lastDir;
    return roadDirArr;
}
```

3.5.2 Tower placement

The code forces towers to be placed exclusively on grass tiles, preventing placement on other types like water or roads. This ensures that tower placement adheres to predefined gameplay rules.

```
private boolean isTileGrass(int x, int y) {
    int id = lvl[y / 32][x / 32];
    int tileType = game.getTileManager().getTile(id).getTileType();
    return tileType == GRASS_TILE;
}

@Override
public void mouseClicked(int x, int y) {
    // Below 640y
    if (y >= 640)
        actionBar.mouseClicked(x, y);
    else {
        // Above 640y
        if (selectedTower != null) {
            // Trying to place a tower
            if (isTileGrass(mouseX, mouseY)) {
                if (getTowerAt(mouseX, mouseY) == null) {
                    towerManager.addTower(selectedTower, mouseX, mouseY);

                    removeGold(selectedTower.getTowerType());

                    selectedTower = null;

                }
            }
        } else {
            // Not trying to place a tower
            // Checking if a tower exists at x,y
            Tower t = getTowerAt(mouseX, mouseY);
            actionBar.displayTower(t);
        }
    }
}
```

3.5.3 Tower Attack Mechanics

If balloons are within the range of a tower, the tower will shoot to destroy them.

```
private void attackEnemyIfClose(Tower t) {
    for (Enemy e : playing.getEnemyManager().getEnemies()) {
        if (e.isAlive())
            if (isEnemyInRange(t, e)) {
                if (t.isCoolDownOver()) {
                    playing.shootEnemy(t, e);
                    t.resetCooldown();
                }
            } else {
                // we do nothing

            }
    }

}
```

3.5.4 Rewards

After destroying balloons, the player earns gold ranging from 3 to 10, depending on the type of balloons they kill.

```
public void addGold(int getReward) {
    this.gold += getReward;

}

public void rewardPlayer(int enemyType) {
    actionBar.addGold(helpz.Constants.Enemies.GetReward(enemyType));
}
```

Moreover, the player also earns 1 gold every 5 seconds while defending.

```
public void update() {
    if (!gamePaused) {
        updateTick();
        waveManager.update();

        // Gold tick
        goldTick++;
        if (goldTick % (60 * 3) == 0)
            actionBar.addGold(1);
```

Chap 4: Final game product

Source code (link github):
https://github.com/enemkayy/BLOONS-TOWER-DEFENSE-PROJECT-OOP_FINAL-

Demo video:
https://www.youtube.com/watch?v=7K_7O2LvI80

4.1 List of objects:

4.1.1 List of enemies:

- Red balloon: The red balloon has 10 HP, the lowest speed, and rewards the player with 3 gold when destroyed.

- Blue balloon: The blue balloon has 20 HP, moves faster than the red balloon, and rewards the player with 5 gold when destroyed.

- Purple balloon: The purple balloon has 30 HP, moves faster than purple balloon, and rewards the player with 8 gold when destroyed.

- Special balloon: The special balloon has 40 HP, the fastest speed, and rewards the player with 10 gold when destroyed.

4.1.2 List of towers:

- Dart Monkey: The dart monkey uses a dart as its weapon, which deals 5 damage, has a range of 100, and the fastest cooldown. After two time upgrades, its maximum damage increases to 10, and its range extends to 120. It costs 30 gold


- Ninja Monkey: The dart monkey uses a shuriken as its weapon, which deals 5 damage, has a range of 100, and a slower cooldown than the dart. Additionally, it has the ability to slow down balloon movement. After two time upgrades, its maximum damage increases to 10, and its range extends to 120. It costs 45 gold


- Super Monkey: The super monkey uses a bomb as its weapon, which deals 10 damage, has a range of 100, and the slowest cooldown. Additionally, it can deal area-of-effect (AoE) damage to multiple balloons at once. After two time upgrades, its maximum damage increases to 20, but its range extends to 120. It costs 60 gold


4.2 User Interface

4.2.1 Main menu:

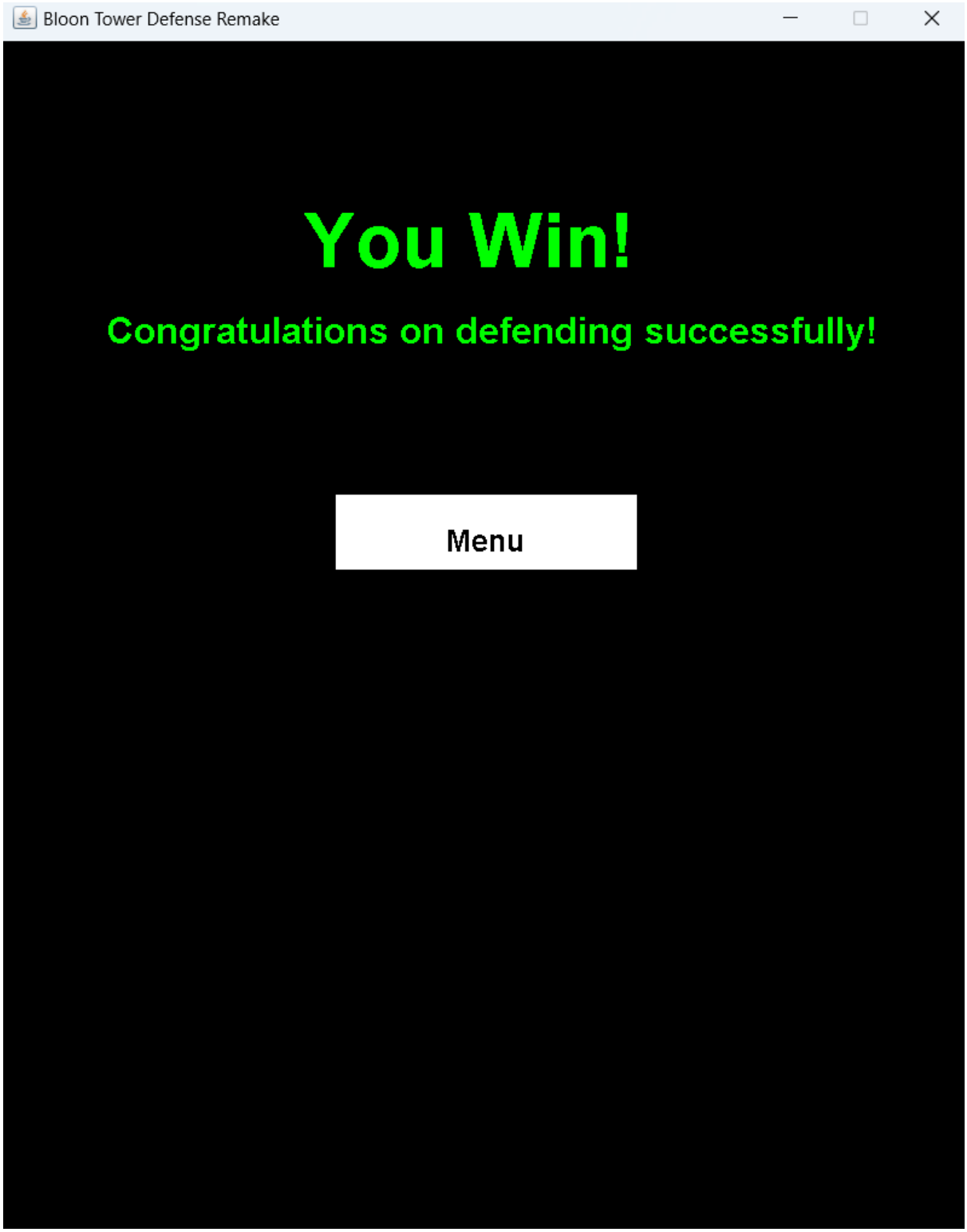
4.2.2 In-Game Screen:

At the beginning game:

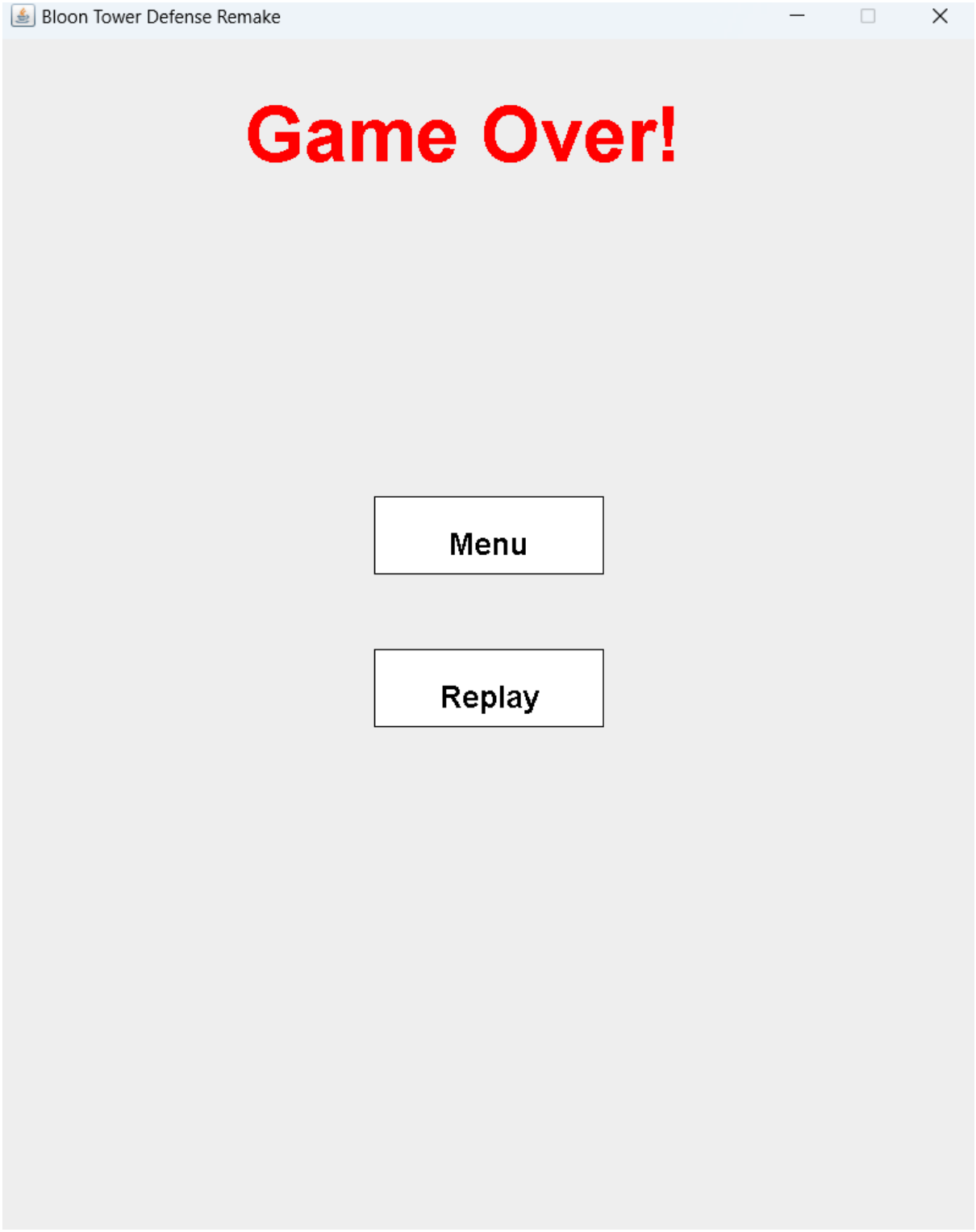
If you click the pause button:

After defending 3 waves of bloons:

if you defend successfully after 10 waves, it will appear win game screen:



or you fail to defense and your health go down 0, it will appear game over screen:


Bloon Tower Defense Remake
Game Over!
Menu
Replay

Chap 5: Experience

This project was an exciting experiment to learn object-oriented programming (OOP). We focused on designing the levels and game mechanics to explore how players would interact with the game. The main feature of the game was programming the monkeys (towers) to pop balloons (enemies) with different behaviors, including attack speed, upgrade paths, and attack range. Balancing the difficulty while keeping the content limited was a challenge, ensuring the game remained engaging without becoming too frustrating.

In addition to the core mechanics, the game included features such as monkey upgrades to improve attack range and abilities. When a level is won, a victory screen is displayed, and if the player loses, a defeat screen appears, giving feedback on the outcome. These elements helped enhance the player's experience and provided a sense of achievement or challenge. Despite the simplicity of the design, debugging and performance optimization were essential challenges, but the smooth gameplay and the achievement of our learning goals made the project rewarding.

In the future, we plan to add several new features to further improve the gaming experience. These will include background music and sound effects, new levels to increase variety, and other interactive elements that will make the game even more engaging. These additions will help make the game more immersive and enjoyable, while continuing to enhance our skills in game development.