VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

PROJECT REPORT

Soul Defender

OBJECT-ORIENTED PROGRAMMING (IT069IU)

Course by

Dr. Tran Thanh Tung & MSc. Nguyen Trung Nghia

Members – Student ID:

Trần Nguyên Khánh - ITDSIU22171

Đặng Huỳnh Minh Phúc - ITITIU22126

Table of contents

CHAPTER 1: INTRODUCTION

1. **Overview:**

- Today, object-oriented programming is one of the most important ways of constructing software. It supports the developer to construct code that is most reusable, usable, and maintainable. In the given application, Java is used to implement conceptualization and demonstrate the different concepts and advantages of OOP for a strong application.
- The project will implement the concepts of OOP: inheritance, encapsulation, polymorphism, and abstraction in real-life problem-solving methodology.

2. **Objective:**

- The purpose of making a "game" is to be proficient in Java programming and the concept of OOP but in a more "gaming" and happier way. Along with that, the project's objectives are to try to improve Java skills and make a stable game structure by using the OOP principle. About the gameplay, it is a tower defense game, in which we defeat the attack waves of enemies while using a limited amount of money to build a unit called tower as allies with their cost.

3. **Used tools:**

- IDE for programming: JetBrains IntelliJ, Eclipse IDE, Visual Code Studio
- Communication: Zalo, Messenger, Discord
- Character, assets design: Aseprite

CHAPTER 2:

METHODOLOGY

1. **Goals:**

Here are our general goals to create a simple tower defense game:

- Variety of Tower and Enemy Types: Offer various types of towers with different attacks and enemies of other kinds acting differently.
- Fun of the game: The game should be entertaining, challenging, and interesting. The game must be one that keeps your guests busy on all the levels and challenges.
- Intuitive Interface: The interface should be simple and readable.
- Replay Value: Players are able to edit and create custom level
- Optimization: The game should be lightweight, can run seamlessly on a wide variety of different devices and platforms with minimal bugs or performance issues.

2. **Rules:**

a. Players and Defending Units:

- Build the defending units, we will call them to as “towers” in order to make it simpler to understand, to defense enemies attack waves.
- There are 3 types of “towers”: “canon”, “elf” and “wizard”.
- Players will have to place “towers” carefully with a limited amount of money.
- Each type of tower has its own price and strengths/weaknesses.
- The amount of money will increase by 1 per second.
- Every tower is upgradable to deal more damage.

b. Enemies (Attacking Units):

- There are 4 kinds of enemies including: “monkey”, “bird”, “pharaon” and “turtle”
- Each kind of enemy will have its own statistics
- Always find the shortest way to the end of the path

- 9 attacking waves which have different amounts and kinds of attacking unit were designed with the increasing difficulty

c. Map:

- A default map was developed
- The map can be modified by the player in edit function

3. Instruction and key binds:

a. Instruction:

- Each tower has its own statistic and cost. Remember to calculate your balance.
- Each enemy also has its own statistics. By defeating them, the game will pay players an amount of gold depending on the kind of enemy.
- The enemies always go on the path, so build the towers along the side of the path to defeat them.
- Each enemy goes to the end point, your HP will be minus by 1. When your HP reaches 0, the game will be over.
- The map can be customized along with the players' idea to make the game more interesting. After finished editing, click the "Save" button to save and apply the map to play.

b. Key binds:

- While playing, there are some keys was made to help the game experience be more convenient:
  - SPACE: Pause the game
  - ESC: Go to main menu
  - X: Clear the current selected unit
  - E: Go to edit
- While editing, there is also key:
  - R: Rotate the selected tile

4. Game Design:

a. UI/UX

- UI/UX are considered as essential parts of the game to attract players. In order to make the art and game concept synchronized.
- However, with our limited knowledge & time, we had to use the button that made with code instead of designing and applying it to the game
- Here are the arts applied in our game:

Figure 2a: Main menu background

Figure 2b: Settings background

Figure 2c: Edit toolbar

Figure 2d: Action bar

Figure 2e: Game over screen

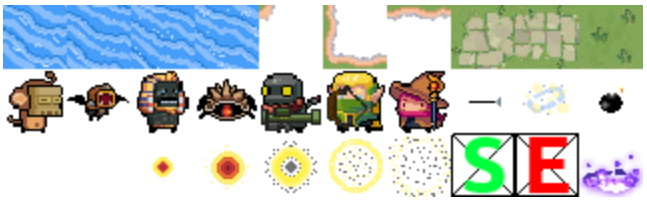
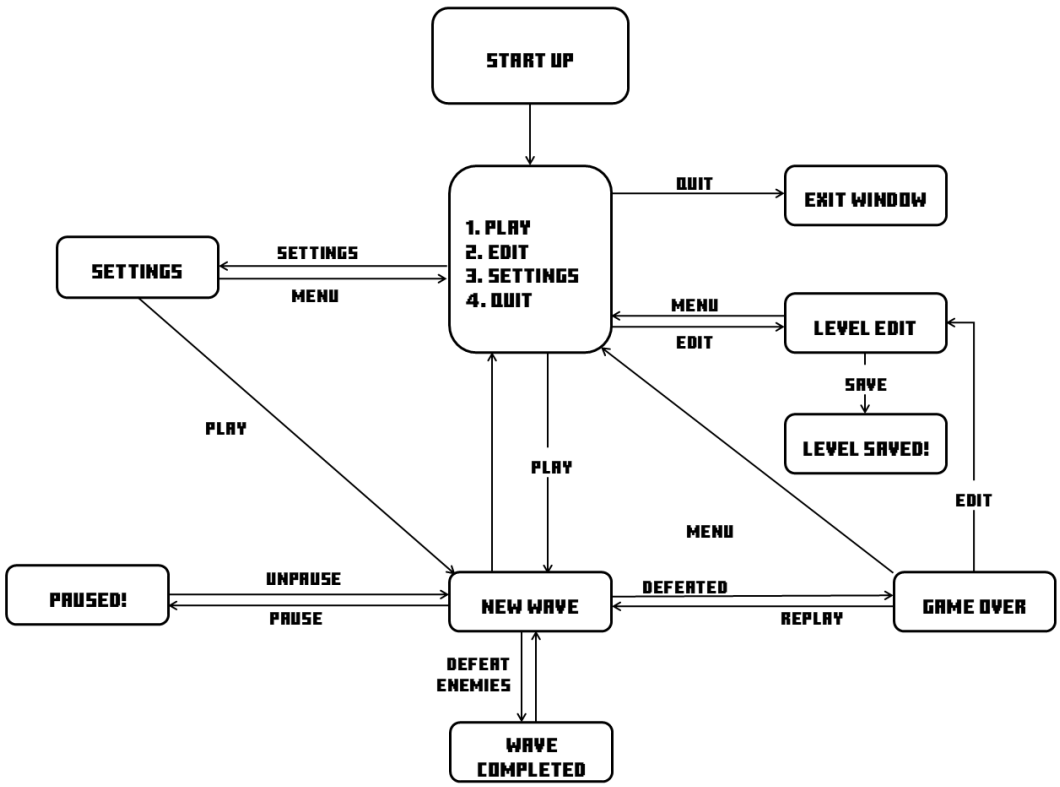


Figure 2f: Sprite alas for tiles, enemies, towers and others

Figure 2g: Winning Logo

b. Game loop:



c.Design pattern:

- State design allows the game to change its behavior according to the internal change. As soon as the object (game) change the current class, the State will do its mission. Here is our code:

```
public enum GameStates {  19 usages

    PLAYING,  10 usages
    MENU,  13 usages
    SETTINGS,  8 usages
    EDIT,  9 usages
    GAME_OVER;  6 usages

    public static GameStates gameState = MENU;  10 usages

    public static void SetGameState(GameStates state) {  10 usages
        gameState = state;
    }

}
```

```
public interface SceneMethods {  5 usages  5 implementations

    public void render(Graphics g);  5 usages  5 implementations

    public void mouseClicked(int x, int y);  5 usages  5 implementations

    public void mouseMoved(int x, int y);  5 usages  5 implementations

    public void mousePressed(int x, int y);  5 usages  5 implementations

    public void mouseReleased(int x, int y);  5 usages  5 implementations

    public void mouseDragged(int x, int y);  4 usages  5 implementations

}
```

```
public class Playing extends GameScene implements SceneMethods {  21 usages
```

```
@Override  5 usages
public void render(Graphics g) {

    drawLevel(g);
    actionBar.draw(g);
    enemyManager.draw(g);
    towerManager.draw(g);
    projManager.draw(g);

    drawSelectedTower(g);
    drawHighlight(g);

}
```

```
public class Menu extends GameScene implements SceneMethods {
```

```
@Override  5 usages
public void render(Graphics g) {

    BufferedImage background = getBackground();
    g.drawImage(background, x: 0, y: 0, width: 640, height: 750, observer: null);
    drawButtons(g);
}
```

```
public class Settings extends GameScene implements SceneMethods {
```

```
@Override  5 usages
public void render(Graphics g) {
    drawBackground(g);
    drawButtons(g);
}
```

```
public class Editing extends GameScene implements SceneMethods {
```

```
@Override  5 usages
public void render(Graphics g) {

    drawLevel(g);
    toolbar.draw(g);
    drawSelectedTile(g);
    drawPathPoints(g);

}
```

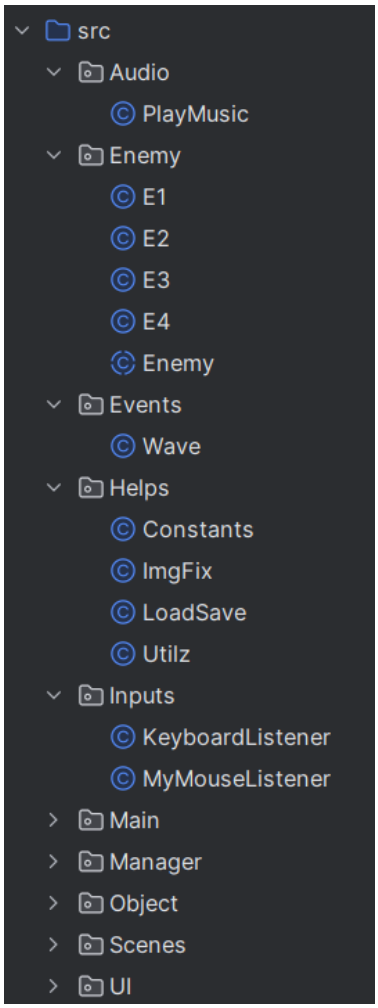
```
public class GameOver extends GameScene implements SceneMethods {
```

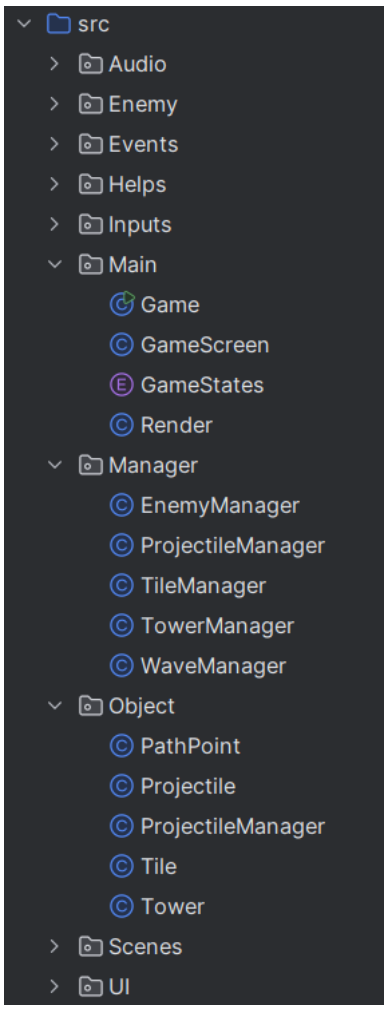
```
@Override  5 usages
public void render(Graphics g) {
    // game over text
    g.setFont(new Font( name: "LucidaSans", Font.BOLD,  size: 50));
    g.setColor(Color.red);
    g.drawString( str: "Game Over!",  x: 160,  y: 80);

    // buttons
    g.setFont(new Font( name: "LucidaSans", Font.BOLD,  size: 20));
    bMenu.draw(g);
    bReplay.draw(g);
}
```

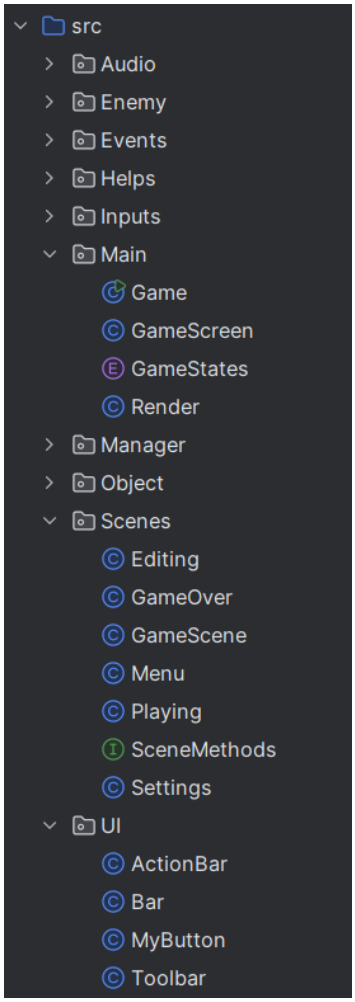
- This project's State class is the abstract class for differently concrete state classes such as Playing, Menu, Edit, Game Over to implement. For example, render(Graphic g) from ScenceMethod gets different implementation methods depending on the different classes, help states will has their own different behavior to the game.

d. Game Algorithm:

- So now, we will come to see the game structure, shown by the image below.







- Main: initializing game screen by extending JPanel and make sure it run stably with Runnable and also make the program run with Game. GameScreen extending JFrame draw on the Window. Gamestate is the Enum class which contains every game states. Render decides what to render depend on running game state.
    - **Game Class**
        - **Game()**: Initializes the game, sets up classes, and creates a default level.
        - **createDefaultLevel()**: Creates a new default level with empty tiles.
        - **initClasses()**: Initializes various game classes like render, menu, playing, etc.
        - **start()**: Starts the game thread.
        - **updateGame()**: Updates the game state based on the current game state.

  - **playSceneMusic(GameStates state)**: Plays background music based on the current game state.
  - **main(String[] args)**: The entry point of the game, creates an instance of the game and starts it.
  - **run()**: The main game loop that handles rendering and updating the game at set frame rates.
  - **getRender()**: Returns the render object.
  - **getMenu()**: Returns the menu object.
  - **getPlaying()**: Returns the playing object.
  - **getSettings()**: Returns the settings object.
  - **getEditor()**: Returns the editing object.
  - **getGameOver()**: Returns the game over object.
  - **getTileManager()**: Returns the tile manager object.
- **GameScreen Class**
  - **GameScreen(Game game)**: Initializes the game screen with a reference to the game.
  - **initInputs()**: Initializes mouse and keyboard input listeners.
  - **setPanelSize()**: Sets the size of the game panel.
  - **paintComponent(Graphics g)**: Paints the game screen, rendering the game graphics.
- **GameStates Class**
  - **SetGameState(GameStates state)**: Sets the current game state to the specified state.
- **Render Class**
  - **Render(Game game)**: Initializes the render class with a reference to the game.
  - **render(Graphics g)**: Renders the current game scene based on the game state.
- **GameScene Class**
  - **GameScene(Game game)**: Initializes the game scene with a reference to the game.
  - **getGame()**: Returns the game object.
  - **isAnimation(int spriteID)**: Checks if the sprite ID corresponds to an animation.
  - **updateTick()**: Updates the animation tick and index for sprite animations.
  - **getSprite(int spriteID)**: Gets the sprite image for the specified ID.

  - **getSprite(int spriteID, int animationIndex)**: Gets the sprite image for the specified ID and animation index.

- **Menu Class**
  - **Menu(Game game)**: Initializes the menu with a reference to the game.
  - **initButtons()**: Initializes the menu buttons.
  - **render(Graphics g)**: Renders the menu on the screen.
  - **getBackground()**: Loads the menu background image.
  - **drawBackground(Graphics g)**: Draws the menu background on the screen.
  - **drawButtons(Graphics g)**: Draws the menu buttons on the screen.
  - **mouseClicked(int x, int y)**: Handles mouse click events on the menu buttons.
  - **mouseMoved(int x, int y)**: Handles mouse movement events to update button states.
  - **mousePressed(int x, int y)**: Handles mouse press events on the menu buttons.
  - **mouseReleased(int x, int y)**: Resets button states on mouse release.
- **Playing Class**
  - **Playing(Game game)**: Initializes the playing scene with a reference to the game.
  - **loadDefaultLevel()**: Loads the default level data.
  - **setLevel(int[][] lvl)**: Sets the current level to the specified level data.
  - **update()**: Updates the game state, including enemies, towers, and projectiles.
  - **isWaveTimerOver()**: Checks if the wave timer is over.
  - **isThereMoreWaves()**: Checks if there are more waves to play.
  - **isAllEnemiesDead()**: Checks if all enemies are dead.
  - **spawnEnemy()**: Spawns a new enemy based on the current wave.
  - **setSelectedTower(Tower selectedTower)**: Sets the currently selected tower.
  - **render(Graphics g)**: Renders the playing scene on the screen.
  - **drawSelectedTower(Graphics g)**: Draws the currently selected tower on the screen.
  - **drawLevel(Graphics g)**: Draws the game level on the screen.

- Scences:
    - ScencesMethods: Interface that contains mouse input methods and render method
    - GameScene: Getting sprite from atlas by getSprite() and running a loop of animation with updatetick()
    - Menu: Draw buttons “Play”, “Edit”, “Settings”, “Quit” and background by @Override the render(). The size and position of buttons are ruled in the initButtons() while background was taken from storage by getBackground().
    - Editing: loadDefaultLevel() will make the game show the default map previously made while building game or made from previous edit. drawLevel(Graphic g) allows players to customize the map as they want with chosen tiles setSelectedTile(Tile tile) which will be placed by changeTile(int x, int y). drawPathPoint(Graphic g) rules starting and ending point for an attacking wave.
    - Playing: loadDefaultLevel() to pick-up the map to render , then choose which tiles that the monsters are able to walk on with getTileType(x,y). Let players choose the tower setSelectedTower() and plant it by drawSelectedTower() and upgrade tower with upgradeTower(), delete it with removerTower(). Check whether any enemy on the field with isAllEnemiesDead(), the program will start a new wave with spawnEnemy(), the waves will be separated in a certain time by isWaveTimerOver(). Beside that, the gold will be added by defeating monsters with rewardPlayer() and reduced by buying with removeGold(). All of these command will be combined to work within update().
    - Settings:
    - GameOver: draw Menu and Replay buttons with render(). With replay buttons, we run the resetAll() to reset every towers planted in previous match and set state as “Playing” again.

- Enemy: abstract class with common attributes: position (x, y), hitbox (bounds), heath, ID, enemyType, direction (lastDir), alive, slowTick (for slow effect cause by towers). These are the methods:
    - EnemyManager: manage enemy's position, ID, type, sets the initial health based on the enemy type and initializes the hitbox
    - setStartHealth(): This private method sets the initial health of the enemy based on its type using a helper method from the Helps.Constants.Enemies class.

- hurt(int dmg): This method reduces the enemy's health by the specified damage amount. If the health drops to zero or below, the enemy is marked as dead, and the player is rewarded for defeating it.
- kill(): This method immediately marks the enemy as dead and sets its health to zero.
- slow(): This method resets the slowTick to apply a slowing effect on the enemy
- move(float speed, int dir): This method updates the enemy's position based on the specified speed and direction. It also checks if the enemy is slowed and adjusts its speed accordingly. The method updates the hitbox position after moving
- updateHitbox(): This private method updates the hitbox's position to match the enemy's current position
- setPos(int x, int y): This method allows setting the enemy's position directly
- getHealthBarFloat(): This method returns the enemy's current health as a percentage of its maximum health, which can be useful for rendering health bars
- Getters: The class includes several getter methods (getX(), getY(), getBounds(), getHealth(), getID(), getEnemyType(), getLastDir(), isAlive(), and isSlowed()) that provide access to the enemy's attributes

- Inputs: manage user interactions, captures keyboard and mouse events, assigns them to the appropriate game components based on the current game state. Here are the methods:
  - keyTyped(KeyEvent e): This method is part of the KeyListener interface but is not used in this implementation. It can be overridden if needed for handling typed key events.
  - keyPressed(KeyEvent e): This method is called when a key is pressed. It checks the current game state and delegates the key press event to the appropriate game component (e.g., the editor or the playing state). This allows different game states to handle input differently.
  - keyReleased(KeyEvent e): This method is also part of the KeyListener interface and is called when a key is released. It is not implemented in this class, but it can be overridden if needed for handling key release events.

- mouseDragged(MouseEvent e): This method is called when the mouse is dragged. It checks the current game state and delegates the mouse drag event to the appropriate game component, allowing for interactions like dragging objects in the editor or gameplay.
- mouseMoved(MouseEvent e): This method is called when the mouse is moved. Similar to the drag method, it checks the current game state and delegates the mouse move event to the appropriate game component.
- mouseClicked(MouseEvent e): This method is called when the mouse is clicked. It can be implemented to handle mouse click events based on the current game state.
- mousePressed(MouseEvent e): This method is called when a mouse button is pressed. It can be implemented to handle mouse press events based on the current game state.
- mouseReleased(MouseEvent e): This method is called when a mouse button is released. It can be implemented to handle mouse release events based on the current game state

- Helps: the class show a pop-up texts to explain purposes of buttons, including these methods:
  - Projectiles: This nested class contains constants related to projectiles, such as their types and methods to get their speed based on the type.
  - ARROW, CHAINS, BOMB: Constants representing different types of projectiles.
  - GetSpeed(int type): A method that returns the speed of a projectile based on its type.
  - Towers: This nested class contains constants related to towers, including their types, costs, names, damage, range, and cooldown.
  - CANNON, ARCHER, WIZARD: Constants representing different types of towers.
  - GetTowerCost(int towerType): Returns the cost of a tower based on its type.
  - GetName(int towerType): Returns the name of a tower based on its type.
  - GetStartDmg(int towerType): Returns the starting damage of a tower based on its type.
  - GetDefaultRange(int towerType): Returns the default range of a tower based on its type.
  - GetDefaultCooldown(int towerType): Returns the default cooldown of a tower based on its type.
  - Direction: This nested class contains constants representing movement directions.
  - LEFT, UP, RIGHT, DOWN: Constants representing the four possible movement directions.

- Enemies: This nested class contains constants related to enemies, including their types, rewards, speeds, and starting health.
- MONKEY, BIRD, PHARAON, TURTLE: Constants representing different types of enemies.
- GetReward(int enemyType): Returns the reward for defeating an enemy based on its type.
- GetSpeed(int enemyType): Returns the speed of an enemy based on its type.
- GetStartHealth(int enemyType): Returns the starting health of an enemy based on its type.
- GetHypoDistance(float x1, float y1, float x2, float y2): This method calculates the Euclidean distance between two points (x1, y1) and (x2, y2) using the Pythagorean theorem, determines distances in the game and checks if an enemy is within range of a tower.

- UI: create user interface components in the game which includes classes for buttons and action bars, for player interactions with its child classed:
  - Bar class: UI elements that represent a rectangular area on the screen (action bar and toolbar). Its method: drawButtonFeedback(Graphics g, MyButton b): feedback for buttons, such as changing their appearance when the mouse hovers over them or when they are pressed.
  - MyButton class: represents a clickable button in the user interface. Methods:
    - draw(Graphics g): Renders the button on the screen, including its body, border, and text.
    - resetBooleans(): Resets the mouse state flags for the button.
    - setMousePressed(boolean mousePressed): Sets the mouse pressed state.
    - setMouseOver(boolean mouseOver): Sets the mouse over state.
    - getBounds(): Returns the bounds of the button for collision detection.
  - ActionBar class extends the Bar class and represents a specific area of the UI that displays game-related information and controls. Methods:
    - draw(Graphics g): Renders the action bar, including buttons, tower information, and player stats (gold and lives).
    - resetEverything(): Resets the action bar's state, including gold and lives.
    - mouseClicked(int x, int y): Handles mouse click events on the action bar and its buttons.
    - mouseMoved(int x, int y): Handles mouse movement events to update button states.

- Toolbar class extends the Bar class and represents a specific area of the user interface dedicated to editing functionalities. Methods:
  - initPathImgs(): Loads the images for the path start and end buttons from the sprite atlas.
  - initButtons(): Initializes the buttons for the toolbar, including the menu, save, start buttons, and tile selection buttons. It populates the map with buttons and their corresponding tile options.
  - saveLevel(): Calls the saveLevel method from the Editing scene to save the current level.
  - rotateSprite(): Rotates the currently selected tile sprite and updates the selected tile accordingly.
  - draw(Graphics g): Renders the toolbar on the screen, including the buttons and any notifications.
  - drawButtons(Graphics g): Renders the individual buttons on the toolbar, including their visual feedback.
  - drawPathButton(Graphics g, MyButton b, BufferedImage img): Renders a path button with its associated image.
  - drawNormalButton(Graphics g, MyButton b): Renders a normal button with its associated image.
  - drawMapButtons(Graphics g): Renders the buttons associated with different tile types.
  - drawSelectedTile(Graphics g): Renders the currently selected tile in the toolbar.
  - mouseClicked(int x, int y): Handles mouse click events on the toolbar buttons, allowing the player to select tiles or perform actions like saving the level.
  - mouseMoved(int x, int y): Updates the state of the buttons based on mouse movement, providing visual feedback for hover states.

- Events:
  - Wave:
    - Wave(ArrayList<Integer> enemyList): Initializes a new wave with a list of enemy types that will spawn during this wave.
    - ArrayList<Integer> getEnemyList(): Returns the list of enemy types for the current wave.
- Manager:
  - WaveManager:

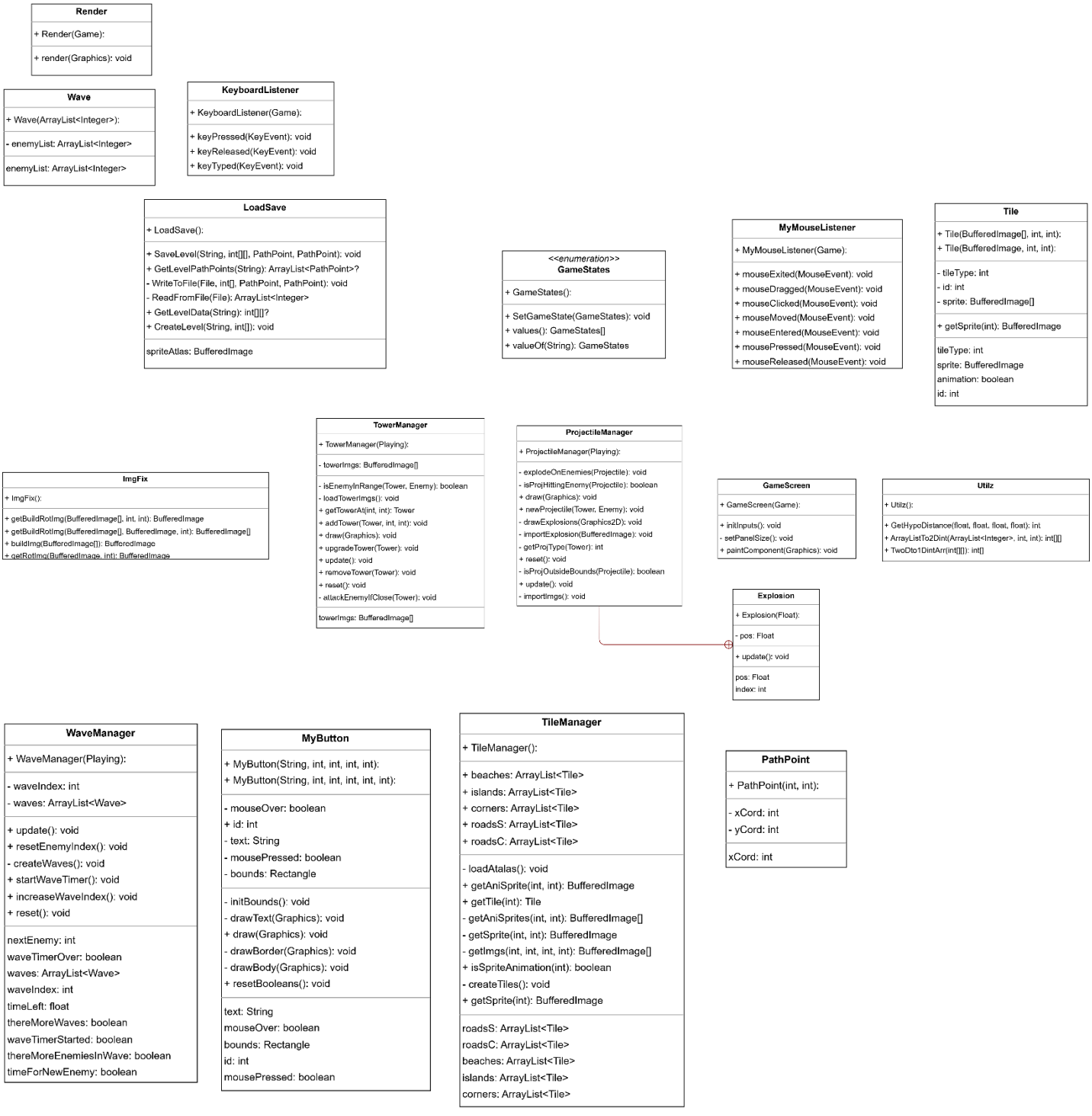
  - WaveManager(Playing playing): Initializes the WaveManager with a reference to the Playing scene and creates the waves of enemies.
  - void update(): Updates the state of the wave manager, including managing the timing for spawning enemies.
  - void increaseWaveIndex(): Increments the index of the current wave and resets the timer for the next wave.
  - boolean isWaveTimerOver(): Checks if the timer for the current wave has expired.
  - void startWaveTimer(): Starts the timer for the current wave.
  - int getNextEnemy(): Returns the next enemy type to spawn from the current wave.
  - void createWaves(): Creates predefined waves of enemies with specific enemy types and counts.
  - ArrayList<Wave> getWaves(): Returns the list of all waves created.
  - boolean isTimeForNewEnemy(): Checks if it's time to spawn a new enemy based on the spawn timer.
  - boolean isThereMoreEnemiesInWave(): Checks if there are more enemies left to spawn in the current wave.
  - boolean isThereMoreWaves(): Checks if there are more waves left to play.
  - void resetEnemyIndex(): Resets the index of the current enemy to spawn in the wave.
  - int getWaveIndex(): Returns the current wave index.
  - float getTimeLeft(): Returns the time left for the current wave timer.
  - boolean isWaveTimerStarted(): Checks if the wave timer has started.
  - void reset(): Resets the wave manager to its initial state, clearing all waves and resetting indices.

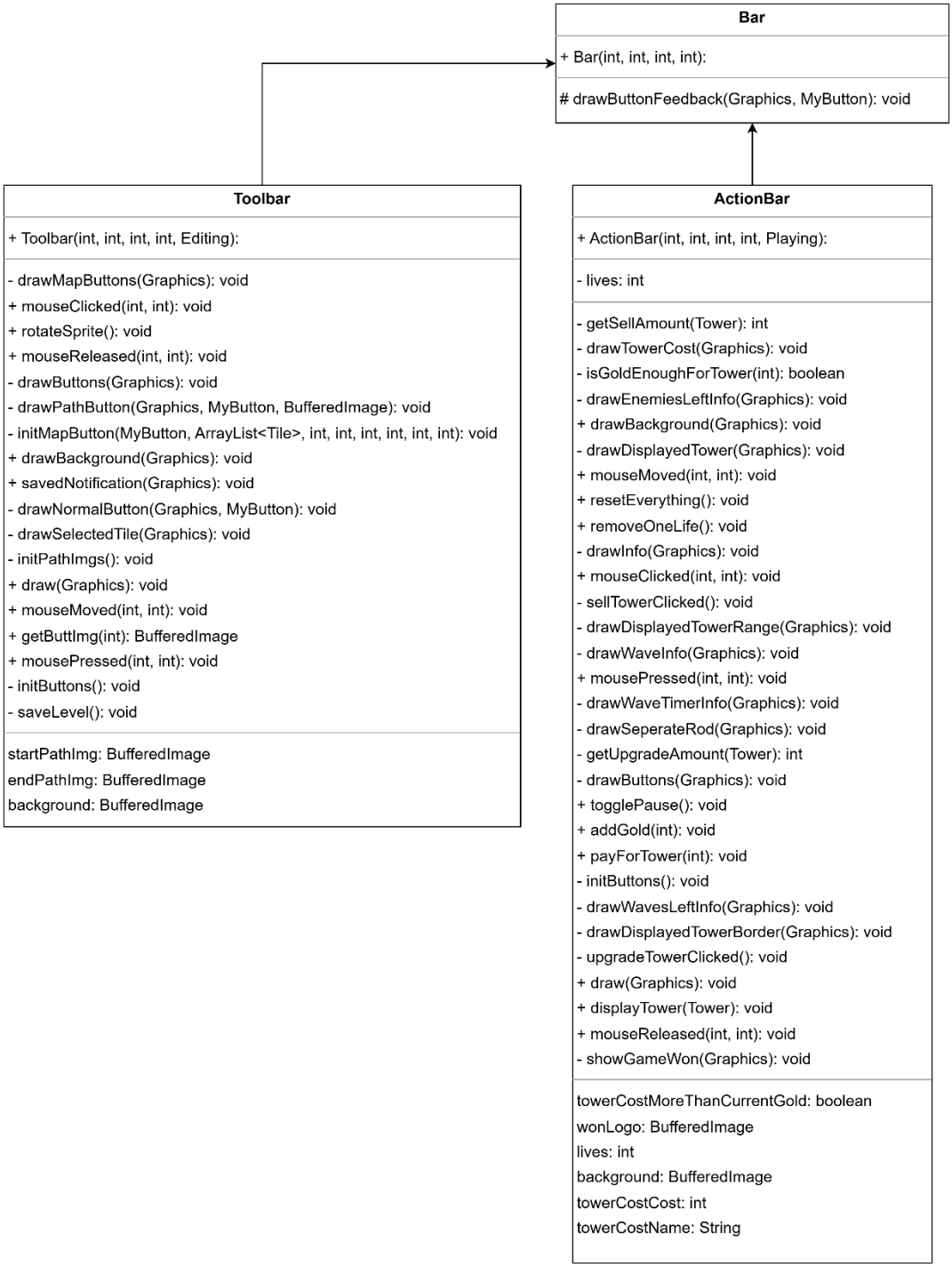
- EnemyManager Class
  - EnemyManager(Playing playing, PathPoint start, PathPoint end): Initializes the enemy manager with a reference to the playing scene and the start and end points for enemy movement.
  - update(): Updates the state of all enemies, moving them if they are alive.
  - updateEnemyMove(Enemy e): Moves a specific enemy based on its last direction and checks if it can move or needs to change direction.

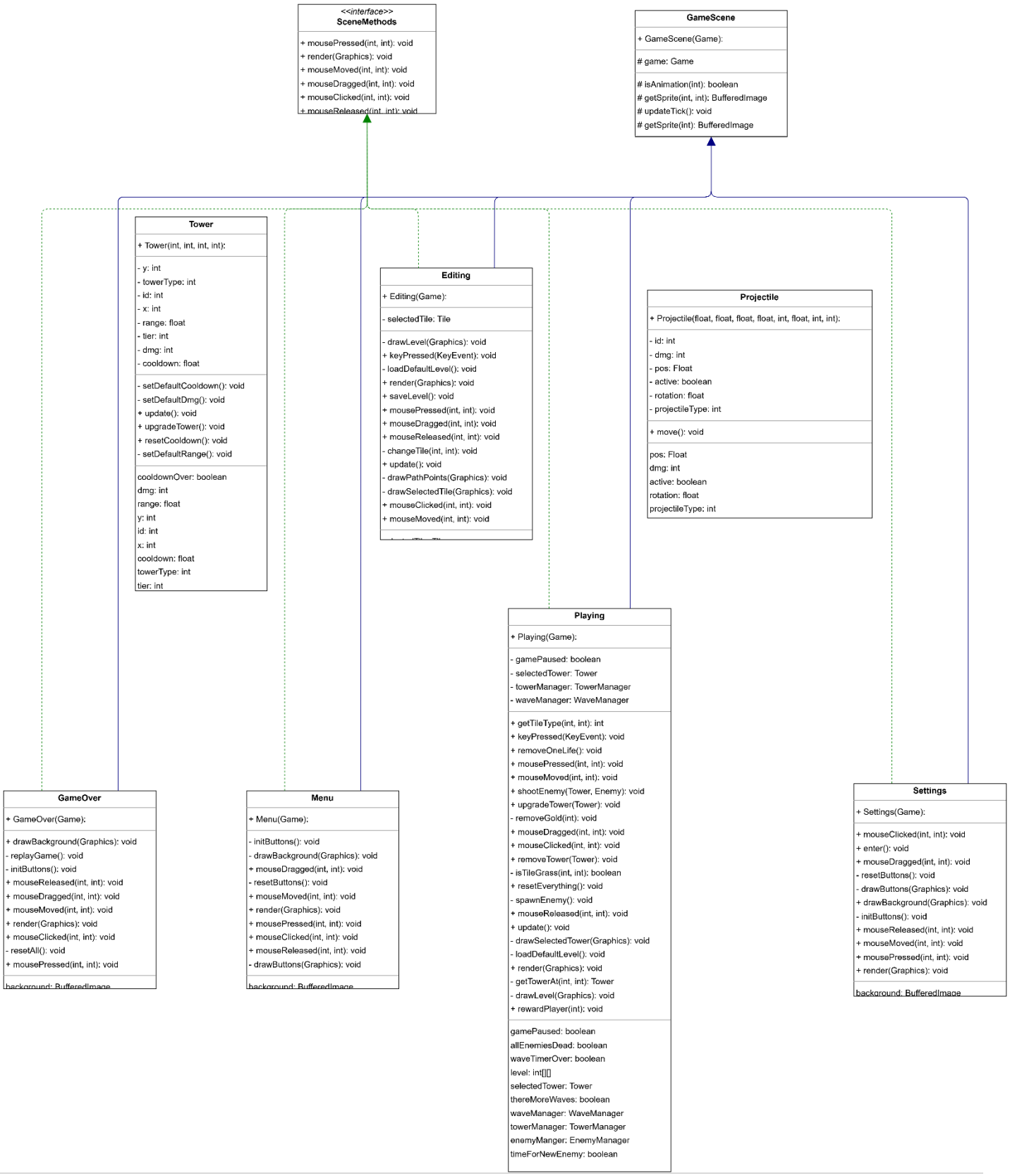
  - setNewDirectionAndMove(Enemy e): Sets a new direction for the enemy to move and updates its position.
  - fixEnemyOffsetTile(Enemy e, int dir, int xCord, int yCord): Adjusts the enemy's position based on its direction and tile coordinates.
  - isAtEnd(Enemy e): Checks if the enemy has reached the end point.
  - getTileType(int x, int y): Gets the type of tile at the specified coordinates.
  - spawnEnemy(int nextEnemy): Spawns a new enemy of the specified type.
  - addEnemy(int enemyType): Adds a new enemy of the specified type to the list.
  - draw(Graphics g): Draws all alive enemies and their health bars on the screen.
  - drawEffects(Enemy e, Graphics g): Draws any effects (like slow effects) on the enemy.
  - drawHealthBar(Enemy e, Graphics g): Draws the health bar for the enemy.
  - getNewBarWidth(Enemy e): Calculates the width of the health bar based on the enemy's health.
  - drawEnemy(Enemy e, Graphics g): Draws the enemy's image on the screen.
  - getEnemies(): Returns the list of enemies.
  - getAmountOfAliveEnemies(): Returns the count of alive enemies.
  - rewardPlayer(int enemyType): Rewards the player based on the type of enemy defeated.
  - reset(): Clears the list of enemies.
- ProjectileManager Class
  - ProjectileManager(Playing playing): Initializes the projectile manager with a reference to the playing scene.
  - importImgs(): Loads the projectile images from the sprite atlas.
  - importExplosion(BufferedImage atlas): Loads explosion images from the sprite atlas.
  - newProjectile(Tower t, Enemy e): Creates a new projectile from a tower towards an enemy.
  - update(): Updates the state of all projectiles and checks for collisions with enemies.
  - explodeOnEnemies(Projectile p): Damages enemies within the explosion radius of a projectile.
  - isProjHittingEnemy(Projectile p): Checks if a projectile is hitting any enemy.

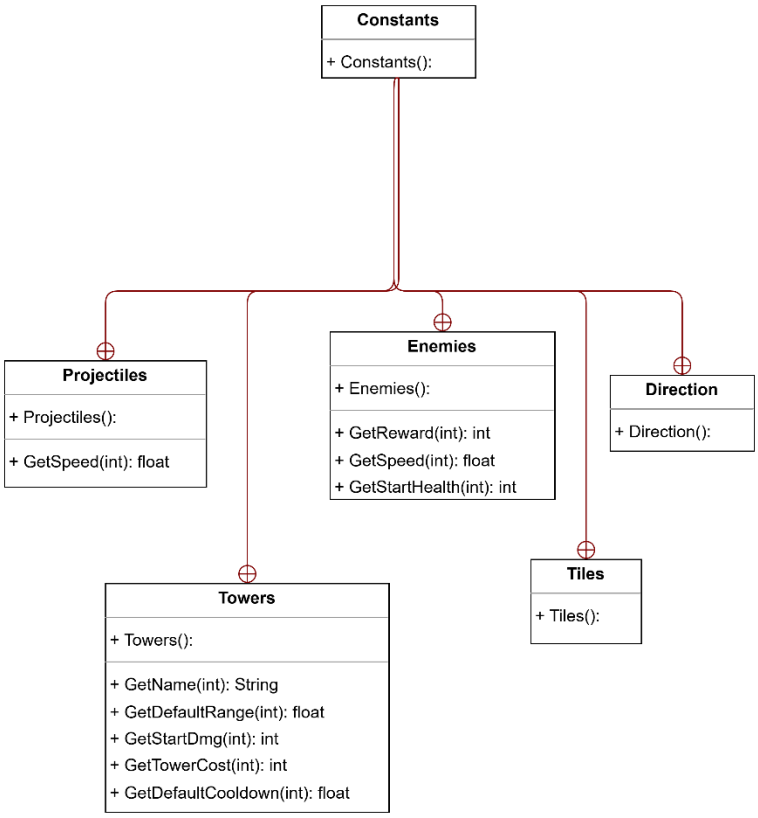
  - isProjOutsideBounds(Projectile p): Checks if a projectile is outside the game bounds.
  - draw(Graphics g): Draws all active projectiles and explosions on the screen.
  - drawExplosions(Graphics2D g2d): Draws all active explosions.
  - getProjType(Tower t): Returns the type of projectile based on the tower type.
  - reset(): Clears the list of projectiles and explosions.
- TowerManager Class
  - TowerManager(Playing playing): Initializes the tower manager with a reference to the playing scene.
  - loadTowerImgs(): Loads tower images from the sprite atlas.
  - addTower(Tower selectedTower, int xPos, int yPos): Adds a new tower at the specified position.
  - removeTower(Tower displayedTower): Removes a specified tower from the list.
  - upgradeTower(Tower displayedTower): Upgrades a specified tower.
  - update(): Updates the state of all towers and checks for enemy attacks.
  - attackEnemyIfClose(Tower t): Checks if any enemy is within range of a tower and attacks if possible.
  - isEnemyInRange(Tower t, Enemy e): Checks if a specific enemy is within the range of a tower.
  - draw(Graphics g): Draws all towers on the screen.
  - getTowerAt(int x, int y): Returns the tower at the specified coordinates.
  - getTowerImgs(): Returns the array of tower images.
  - reset(): Clears the list of towers.

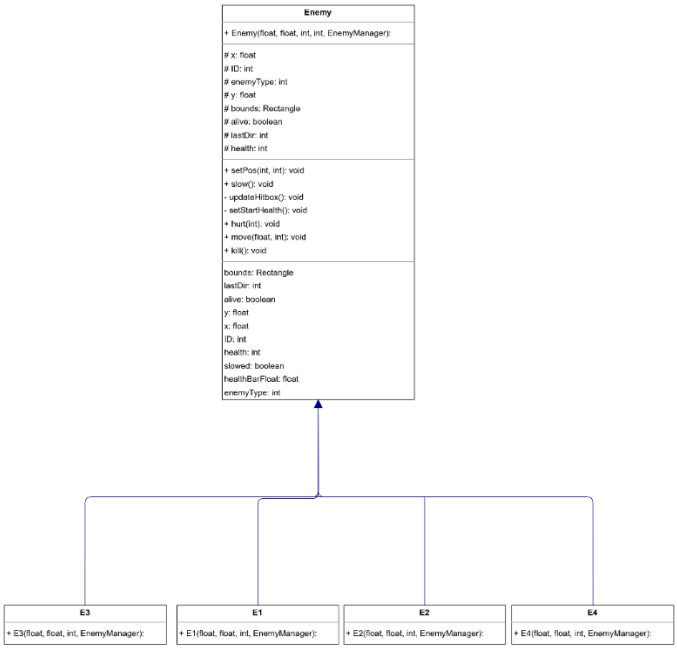
5. UML Diagram:

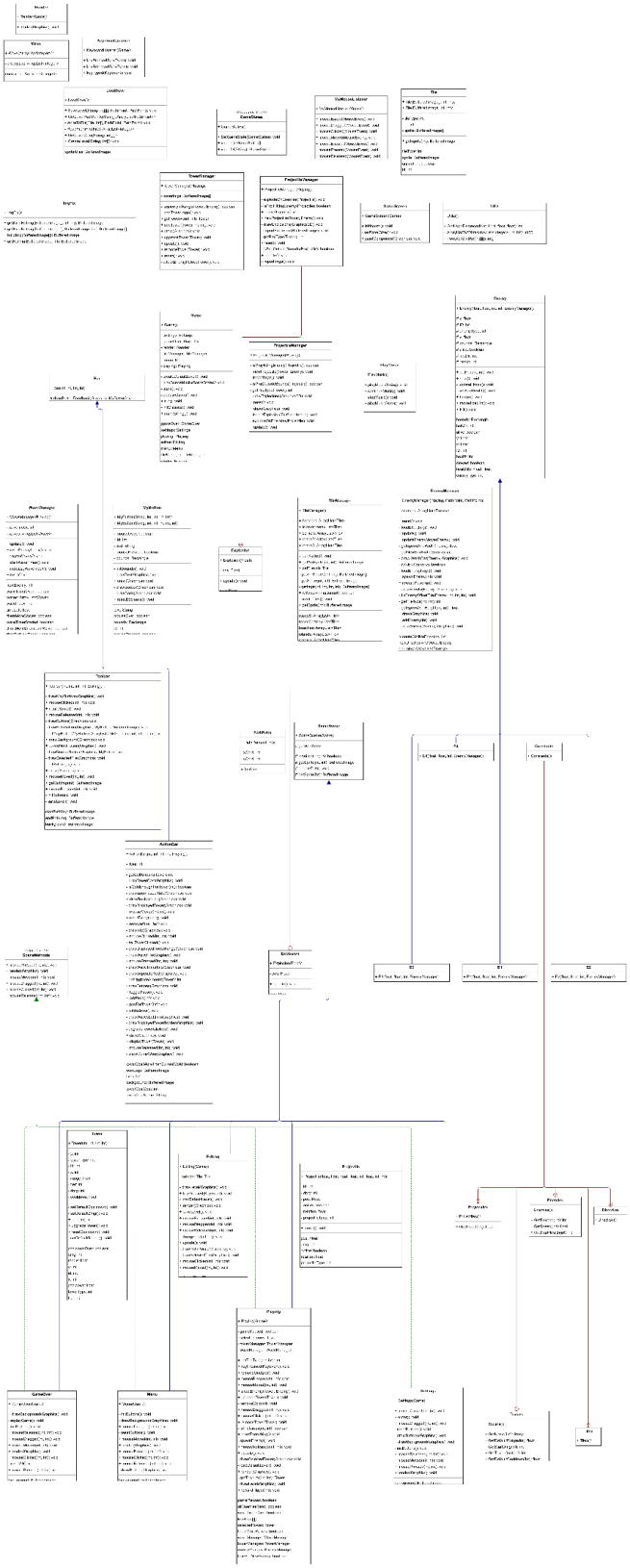



Bar
+ Bar(int, int, int, int):
# drawButtonFeedback(Graphics, MyButton): void
Toolbar
+ Toolbar(int, int, int, int, Editing):
- drawMapButtons(Graphics): void
+ mouseClicked(int, int): void
+ rotateSprite(): void
+ mouseReleased(int, int): void
- drawButtons(Graphics): void
- drawPathButton(Graphics, MyButton, BufferedImage): void
- initMapButton(MyButton, ArrayList<Tile>, int, int, int, int, int, int): void
+ drawBackground(Graphics): void
+ savedNotification(Graphics): void
- drawNormalButton(Graphics, MyButton): void
- drawSelectedTile(Graphics): void
- initPathImgs(): void
+ draw(Graphics): void
+ mouseMoved(int, int): void
+ getButtImg(int): BufferedImage
+ mousePressed(int, int): void
- initButtons(): void
- saveLevel(): void
startPathImg: BufferedImage
endPathImg: BufferedImage
background: BufferedImage
ActionBar
+ ActionBar(int, int, int, int, Playing):
- lives: int
- getSellAmount(Tower): int
- drawTowerCost(Graphics): void
- isGoldEnoughForTower(int): boolean
- drawEnemiesLeftInfo(Graphics): void
+ drawBackground(Graphics): void
- drawDisplayedTower(Graphics): void
+ mouseMoved(int, int): void
+ resetEverything(): void
+ removeOneLife(): void
- drawInfo(Graphics): void
+ mouseClicked(int, int): void
- sellTowerClicked(): void
- drawDisplayedTowerRange(Graphics): void
- drawWaveInfo(Graphics): void
+ mousePressed(int, int): void
- drawWaveTimerInfo(Graphics): void
- drawSeperateRod(Graphics): void
- getUpgradeAmount(Tower): int
- drawButtons(Graphics): void
+ togglePause(): void
+ addGold(int): void
+ payForTower(int): void
- initButtons(): void
- drawWavesLeftInfo(Graphics): void
- drawDisplayedTowerBorder(Graphics): void
- upgradeTowerClicked(): void
+ draw(Graphics): void
+ displayTower(Tower): void
+ mouseReleased(int, int): void
- showGameWon(Graphics): void
towerCostMoreThanCurrentGold: boolean
wonLogo: BufferedImage
lives: int
background: BufferedImage
towerCostCost: int
towerCostName: String


<<interface>>
SceneMethods
+ mousePressed(int, int): void
+ render(Graphics): void
+ mouseMoved(int, int): void
+ mouseDragged(int, int): void
+ mouseClicked(int, int): void
+ mouseReleased(int, int): void
GameScene
+ GameScene(Game):
# game: Game
# isAnimation(int): boolean
# getSprite(int, int): BufferedImage
# updateTick(): void
# getSprite(int): BufferedImage
Tower
+ Tower(int, int, int, int):
- y: int
- towerType: int
- id: int
- x: int
- range: float
- tier: int
- dmg: int
- cooldown: float
- setDefaultCooldown(): void
- setDefaultDmg(): void
+ update(): void
+ upgradeTower(): void
+ resetCooldown(): void
- setDefaultRange(): void
cooldownOver: boolean
dmg: int
range: float
y: int
id: int
x: int
cooldown: float
towerType: int
tier: int
Editing
+ Editing(Game):
- selectedTile: Tile
- drawLevel(Graphics): void
+ keyPressed(KeyEvent): void
- loadDefaultLevel(): void
+ render(Graphics): void
+ saveLevel(): void
+ mousePressed(int, int): void
+ mouseDragged(int, int): void
+ mouseReleased(int, int): void
- changeTile(int, int): void
+ update(): void
- drawPathPoints(Graphics): void
- drawSelectedTile(Graphics): void
+ mouseClicked(int, int): void
+ mouseMoved(int, int): void
Projectile
+ Projectile(float, float, float, float, int, float, int, int):
- id: int
- dmg: int
- pos: Float
- active: boolean
- rotation: float
- projectileType: int
+ move(): void
pos: Float
dmg: int
active: boolean
rotation: float
projectileType: int
Playing
+ Playing(Game):
- gamePaused: boolean
- selectedTower: Tower
- towerManager: TowerManager
- waveManager: WaveManager
+ getTileType(int, int): int
+ keyPressed(KeyEvent): void
+ removeOneLife(): void
+ mousePressed(int, int): void
+ mouseMoved(int, int): void
+ shootEnemy(Tower, Enemy): void
+ upgradeTower(Tower): void
- removeGold(int): void
+ mouseDragged(int, int): void
+ mouseClicked(int, int): void
+ removeTower(Tower): void
- isTileGrass(int, int): boolean
+ resetEverything(): void
- spawnEnemy(): void
+ mouseReleased(int, int): void
+ update(): void
- drawSelectedTower(Graphics): void
- loadDefaultLevel(): void
+ render(Graphics): void
- getTowerAt(int, int): Tower
- drawLevel(Graphics): void
+ rewardPlayer(int): void
gamePaused: boolean
allEnemiesDead: boolean
waveTimerOver: boolean
level: int[][]
selectedTower: Tower
thereMoreWaves: boolean
waveManager: WaveManager
towerManager: TowerManager
enemyManger: EnemyManager
timeForNewEnemy: boolean
GameOver
+ GameOver(Game):
+ drawBackground(Graphics): void
- replayGame(): void
- initButtons(): void
+ mouseReleased(int, int): void
+ mouseDragged(int, int): void
+ mouseMoved(int, int): void
+ render(Graphics): void
+ mouseClicked(int, int): void
- resetAll(): void
+ mousePressed(int, int): void
background: BufferedImage
Menu
+ Menu(Game):
- initButtons(): void
- drawBackground(Graphics): void
+ mouseDragged(int, int): void
- resetButtons(): void
+ mouseMoved(int, int): void
+ render(Graphics): void
+ mousePressed(int, int): void
+ mouseClicked(int, int): void
+ mouseReleased(int, int): void
- drawButtons(Graphics): void
background: BufferedImage
Settings
+ Settings(Game):
+ mouseClicked(int, int): void
+ enter(): void
+ mouseDragged(int, int): void
- resetButtons(): void
- drawButtons(Graphics): void
+ drawBackground(Graphics): void
- initButtons(): void
+ mouseReleased(int, int): void
+ mouseMoved(int, int): void
+ mousePressed(int, int): void
+ render(Graphics): void
background: BufferedImage


Constants
+ Constants():
Projectiles
+ Projectiles():
+ GetSpeed(int): float
Enemies
+ Enemies():
+ GetReward(int): int
+ GetSpeed(int): float
+ GetStartHealth(int): int
Direction
+ Direction():
Towers
+ Towers():
+ GetName(int): String
+ GetDefaultRange(int): float
+ GetStartDmg(int): int
+ GetTowerCost(int): int
+ GetDefaultCooldown(int): float
Tiles
+ Tiles():


Enemy
+ Enemy(float, float, int, int, EnemyManager):
# x: float
# ID: int
# enemyType: int
# y: float
# bounds: Rectangle
# alive: boolean
# lastDir: int
# health: int
+ setPos(int, int): void
+ slow(): void
- updateHitbox(): void
- setStartHealth(): void
+ hurt(int): void
+ move(float, int): void
+ kill(): void
bounds: Rectangle
lastDir: int
alive: boolean
y: float
x: float
ID: int
health: int
slowed: boolean
healthBarFloat: float
enemyType: int
E3
+ E3(float, float, int, EnemyManager):
E1
+ E1(float, float, int, EnemyManager):
E2
+ E2(float, float, int, EnemyManager):
E4
+ E4(float, float, int, EnemyManager):



CHAPTER 3:
RESULT

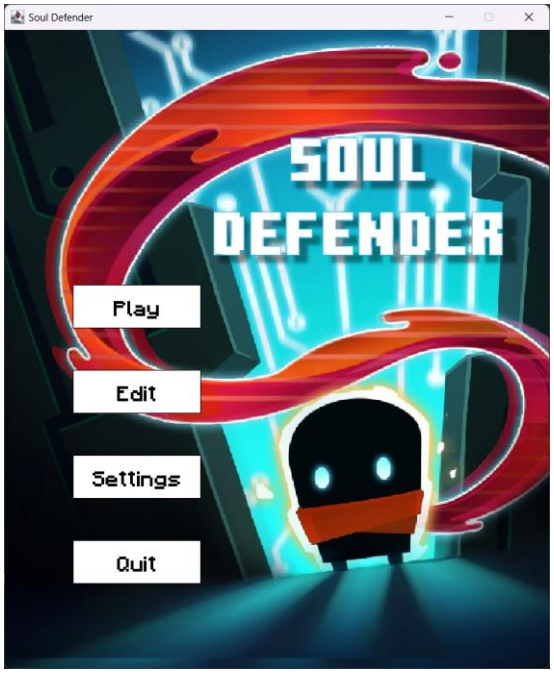


Figure 3a Menu

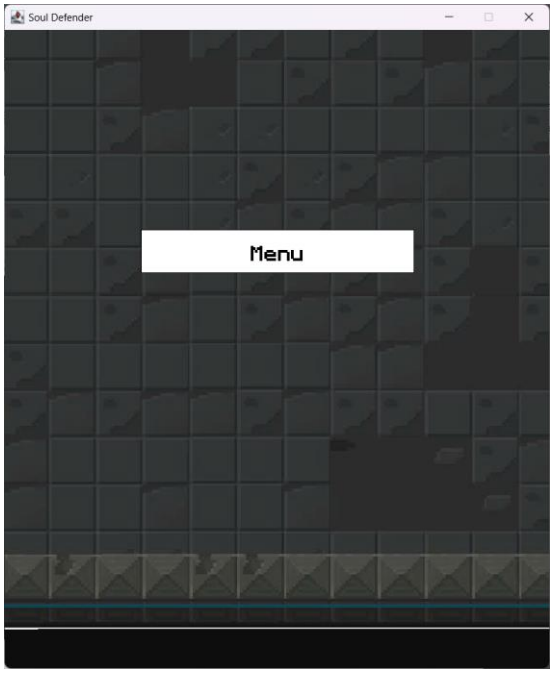
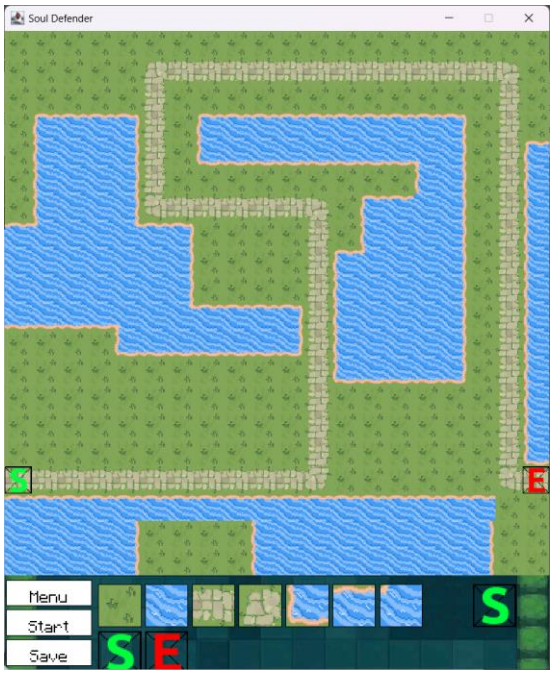
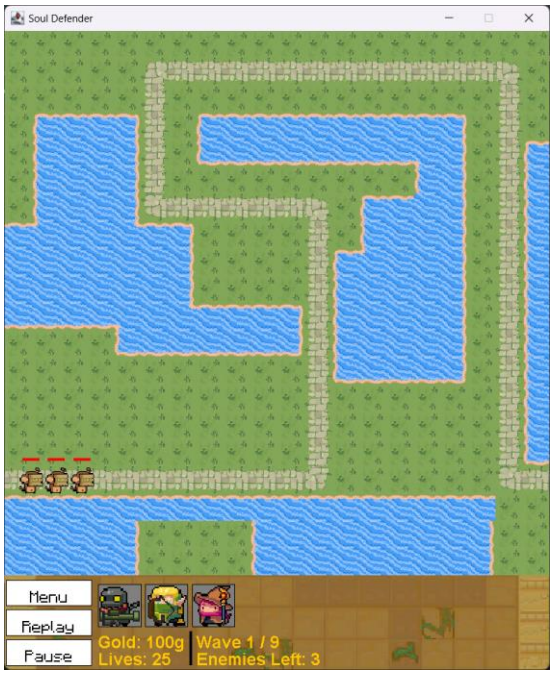


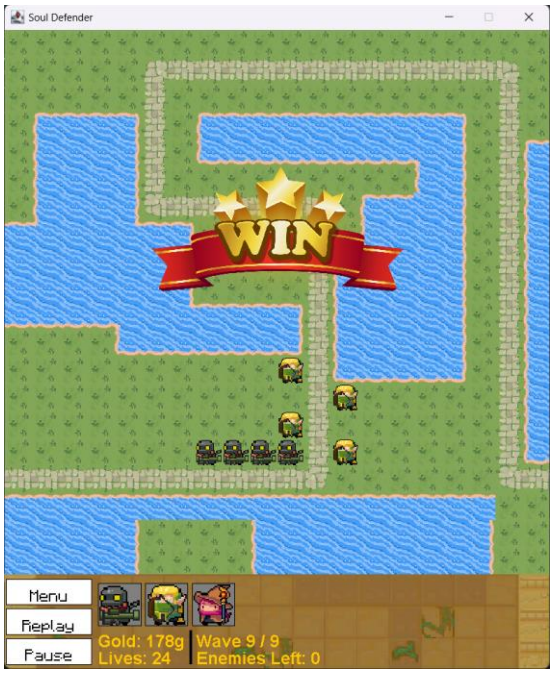
Figure 3b Settings



3c Editing



3d Playing



3e Winning Scene

3f Losing Scene

CHAPTER 4: CONCLUSION

1. **Summary:**
   - During the session of making this game, we have learnt several of things. Firstly, we gained a deeper knowledge of four core concept of OOP including inheritance, polymorphism, abstraction, and encapsulation. This is helpful for us to acknowledge the way to apply OOP concepts to practical.

2. **Future works:**
   - Beside fixing bugs that is not fixed currently, there are more features that need to be added:
     - The volume customization (mute/unmute, volume setters).
     - Animation for towers, monsters.
     - Options while playing game.
     - Winning scence instead of a notification.

     and more…

3. **Acknowledgement:**

   Our sincere appreciation goes out to our instructor and everyone else who has helped us achieve the project's goals with their tremendous assistance:
     - Dr. Tran Thanh Tung
     - MSc. Nguyen Trung Nghia
     - Original code from Kaarin Gaming

List of reference

Tutorial:

https://www.coursera.org/specializations/java-programming

https://www.w3schools.com/java/

https://codingtechroom.com/tutorial/creating-2d-games-with-java-beginner-tutorial

https://learncodebygaming.com/blog/how-to-make-a-video-game-in-java-2d-basics

https://www.kaaringaming.com/tutorials

https://www.youtube.com/watch?v=JhqyVeEeRsg

https://github.com/basilvetas/tower-defense

Audio:

mainmenu.wav	https://www.youtube.com/watch?v=Wyb91HPNz68
play.wav	https://www.youtube.com/watch?v=hLPvc5mZiCY
edit.wav	https://www.youtube.com/watch?v=vTAUDCazz34
lose.wav	https://www.youtube.com/watch?v=L0W8WwMOxOE

Towers (inspire from):

https://soul-knight.fandom.com/wiki/Characters

https://soul-knight.fandom.com/wiki/Weapons

Enemies (inspire from):

https://soul-knight.fandom.com/wiki/Enemies

Sprites:

https://www.kaaringaming.com/tutorials

Background:

bg (1).png	https://www.facebook.com/chillyroomsoulknight/
play.png	https://soul-knight.fandom.com/wiki/Relics
edit.png	https://soul-knight.fandom.com/wiki/Forest
over.png	* CAPTURE FROM SOULKNIGHT GAMEPLAY FOOTAGE *

Others:

WinLogo.png	https://pngtree.com/so/win-logo