VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

PROJECT REPORT

PLANT VS ZOMBIES IN CITY

OBJECT-ORIENTED PROGRAMMING (IT069IU)

Course by

Dr. Tran Thanh Tung and MSc. Nguyen Trung Nghia

Members_ID Student
Bùi Ngọc Quang Huy_ITDSIU22155
Lê Trọng Hiếu_ITDSIU22127
Tô Duy Thịnh_ITCSIU22138
Nguyễn Trường Minh Quang_ITCSIU22224

Table of Content:

I. Introduction:

1. About own team:

Our team consists of four dedicated members with diverse expertise in software development. Each member brings unique skills in Java programming, software design, and game development, which collectively contribute to the success of this project.

- Our Project: GitHub

2. Objective:

The main goal of making "Plant vs Zombies" is to get better at Java programming while using Object-Oriented Programming (OOP) ideas to create a fun and easy-to-play game. The story of the game is about protecting a house from zombies by planting different types of plants strategically. This project wants to improve Java skills and promote a good way of building games by using OOP principles. Apart from entertaining players from different backgrounds, the game also wants to improve our abilities in game development. Our team is committed to making sure players have a great time and to using our technical skills to make "Plant vs Zombies" a success.
- Goals Summary:

- Make a game that helps with learning and practicing Java.
- Use and improve OOP principles as we work on the game.
- Work on improving the game and managing it as we go.
- See how well we can add new features to the main game.
- Learn more about programming, setting up projects, and working together as a team o a big project with different parts.

3. Tool Used:

- IDE for programming and debugging: VSCode IntelliJ.
- Version Control: Git/GitHub.
- Manage tasks and keep track of tasks: Task Management
- Character/Map design and image editing: Adobe Photoshop, Pixel Studio,Pixilart.
- Sprite sheet Splitting or Sprite Sheet Decomposition: PineTools.

II. Methodology:

1. Game Rule:

a) Description:

The game "Plants vs. Zombies" involves players placing various types of plants to fend off waves of zombies. The objective is to prevent zombies from reaching the player's house by strategically planting defensive and offensive plants.

b) Rules:

In Plants vs.Zombies, players will play the role of a homeowner in the midst of a zombie apocalypse. When a horde of zombies begins to approach the house along parallel lanes, the player must defend the house by planting plants to destroy or disadvantage the zombies. Players collect a currency called "sun" ("sun") to buy plants. If a zombie reaches the house using any lane, the level is considered failed, and the player will have to replay the level.

2. Game Instruction:

a) Start the Game:

- Initial Sun Points:At the beginning of the game, players are provided with a limited number of "sun" points, the game's currency. For example, players might start with 100 sun points.
- Objective:The main objective is to prevent zombies from reaching the player's house at the right side of the lawn by planting various plants that have different defensive and offensive capabilities.

b) Plant Selection:

- Choosing Plants:Players can choose from a variety of plants to place on their lawn. Each plant type requires a different amount of sun points. For example, a Peashooter may cost 100 sun points, while a Sunflower costs 50 sun points.
- Strategy:Players need to consider their strategy and the types of zombies they will face. Some plants are better suited for attacking, while others are best for defense or sun point generation.

c) Plant Placement:

- Placing Plants:Players can place selected plants on any empty square of their lawn grid. The placement is crucial as it determines the plant's effectiveness in defending against zombies.
- Grid Layout:The lawn is divided into a grid, usually 5 rows and 9 columns. Players drag and drop plants onto the grid to position them strategically.

d) Sun Collection:

- Generating Sun Points: Sun points can be collected in multiple ways. Sunflowers generate sun points periodically, and additional sun points fall from the sky at intervals.
- Manual Collection: Players must click on the sun points to collect them, which adds to their total number of sun points available for planting more plants.

e) Wave Management:

- Zombie Waves: Zombies attack in waves, with each subsequent wave being more challenging. Players must manage their plant defenses to handle these increasing threats.
- Interim Periods:Between waves, there may be short periods where players can adjust their strategies, plant additional plants, and collect more sun points.

f) Win/Lose Conditions:

- Winning the Game:The game is won when all waves of zombies are successfully repelled. Players progress through levels or stages as they continue to win.
- Losing the Game:
- The game is lost if zombies manage to breach the player's defenses and reach the house. This usually results in a game over scenario.
- Restarting:
- Players can restart the game from the beginning or from a menu

3. UML:

a) Plant UML:

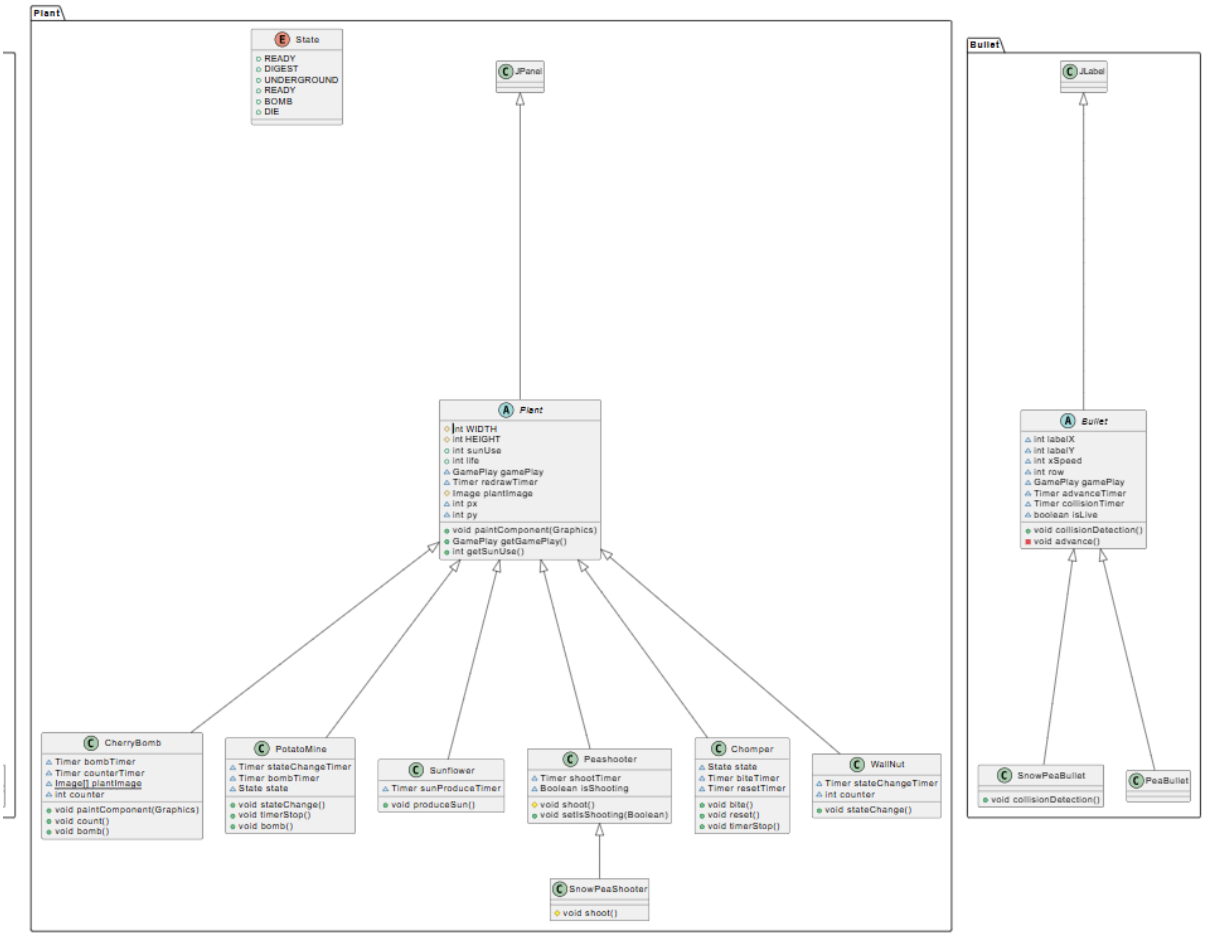


Figure 2a.1: UML of Class Plant

b) Zombies UML:

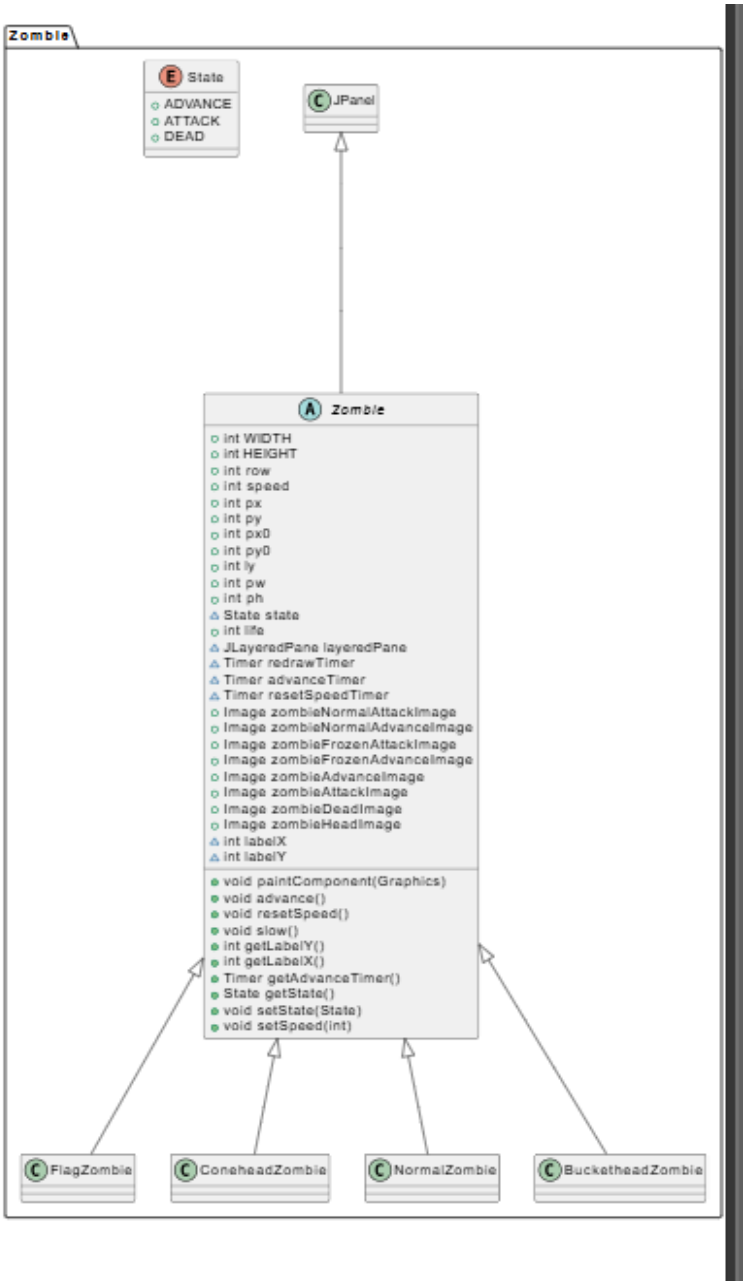


Figure 2a.2: UML of the Zombie

c) Game Control UML:

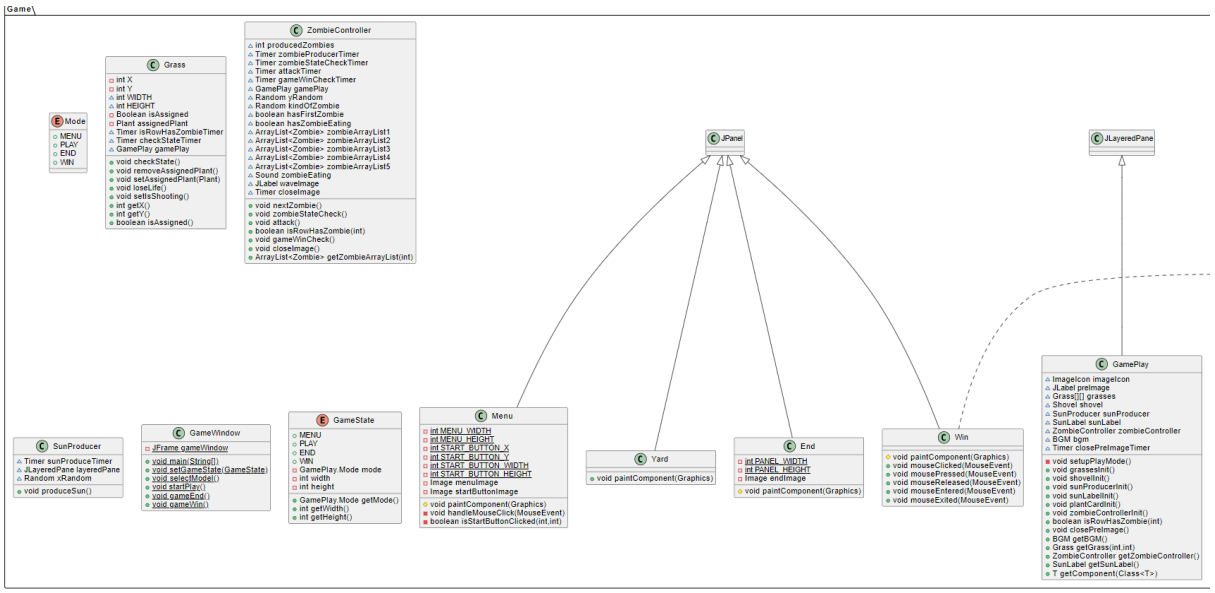


Figure 2a.3: UML of Game Control

d) UML of whole project:

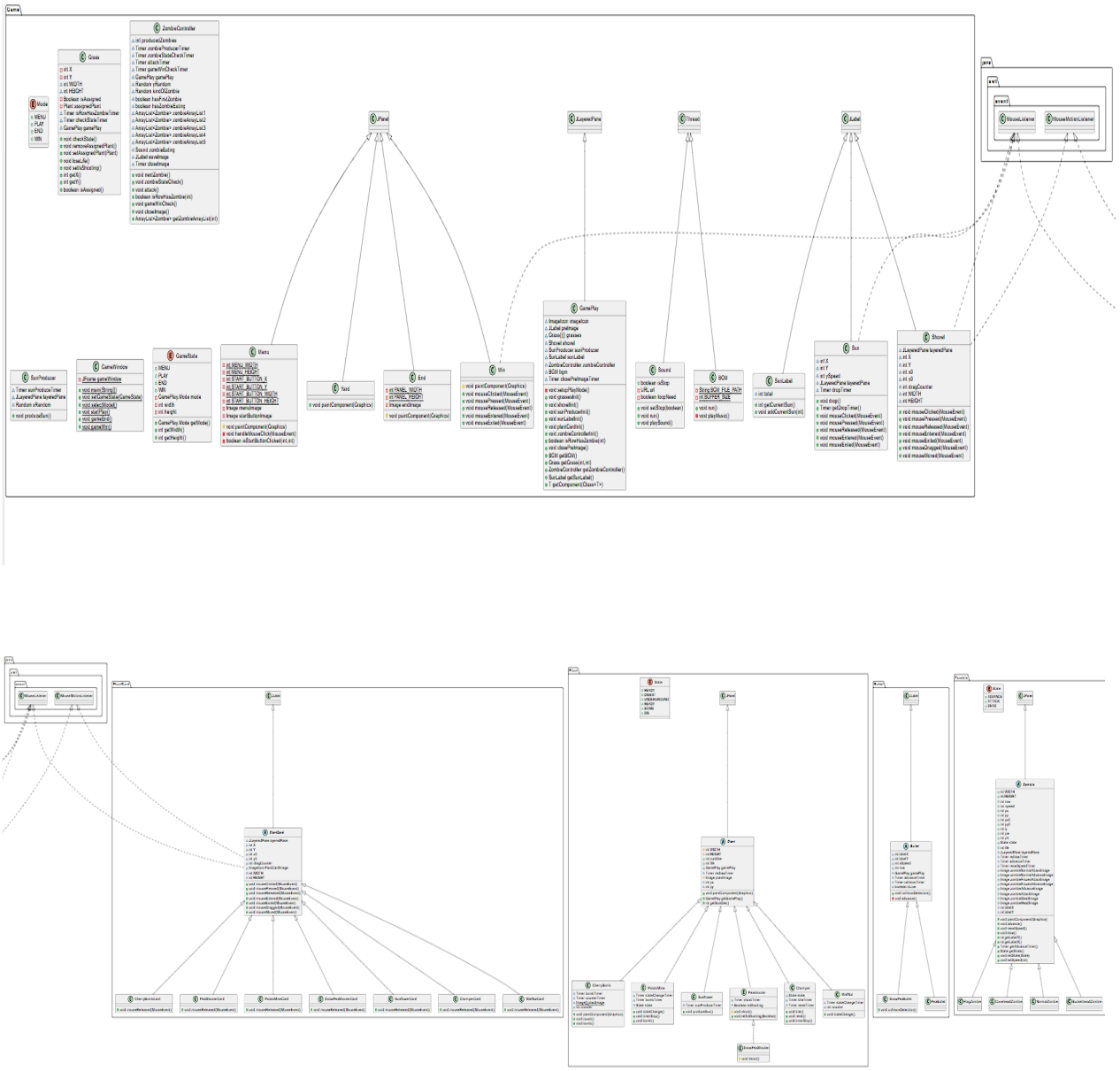
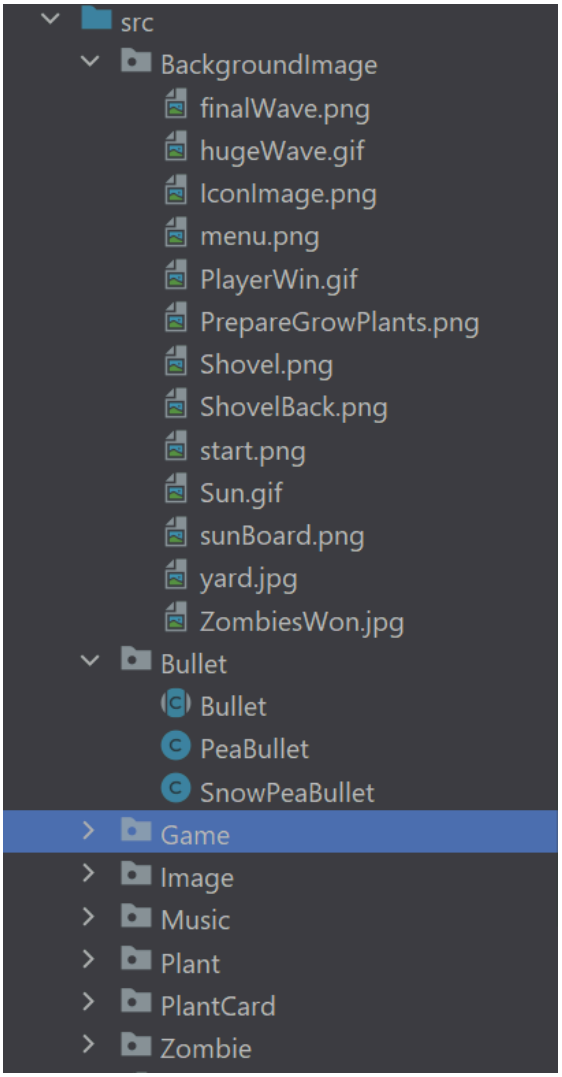
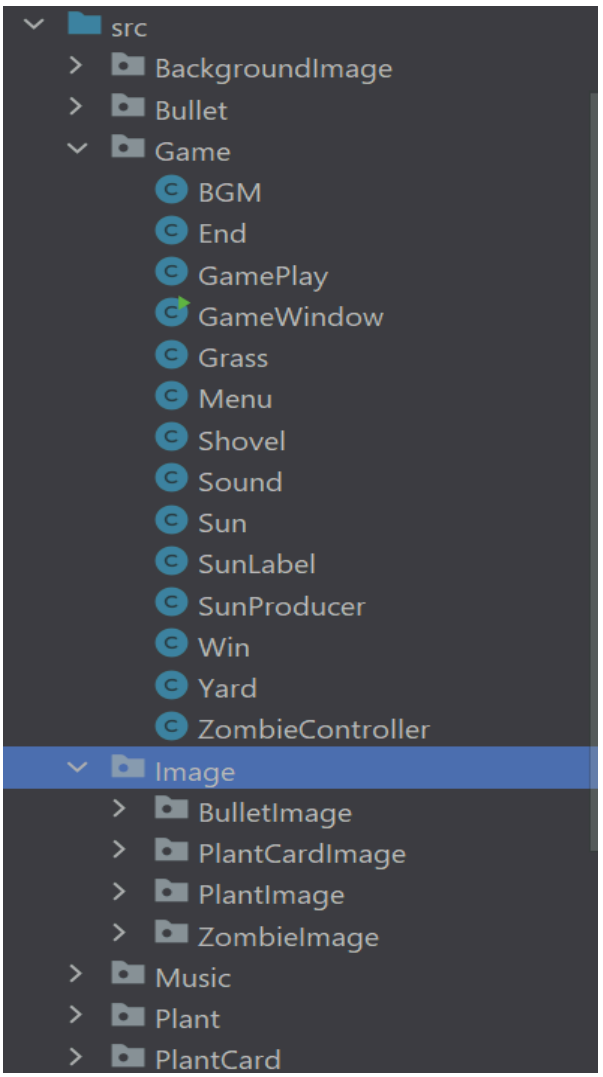
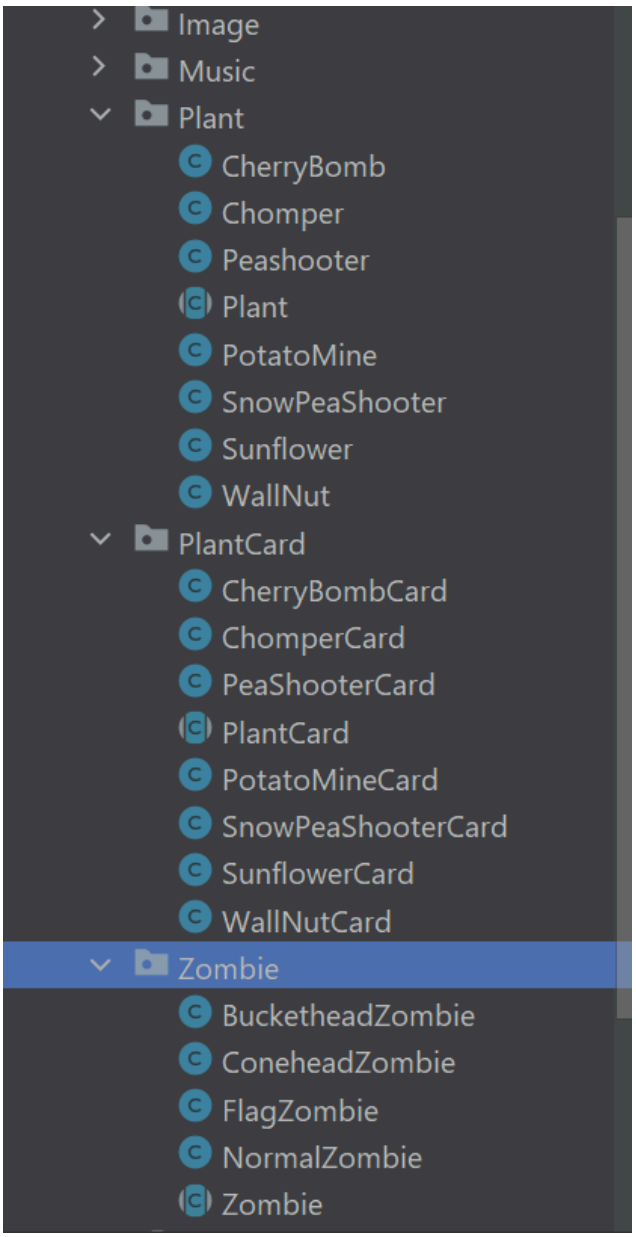


Figure 2a.4: **UML of whole project**

4. Game Algorithm:







- We group our classes into a specific group, such as:
  - BackgroundImage: In this folder, we have the image for the game when we started, like PrepareGrowPlants.png, Shovel.png, ShovelBack.png, ShovelBack.png, Sun.gif, sunBoard.png, yard.jpg, ZombiesWon.jpg.
  - Package Bullet:
  - **Bullet Class (Abstract)**:
  - This class extends JLabel and serves as the base class for all types of bullets in the game.

- It includes attributes such as labelX, labelY for position, xSpeed for bullet speed, row for the row of the bullet, gamePlay for the game instance, and timers for bullet advancement and collision detection.
- The constructor initializes these attributes and starts the timers.
- The collisionDetection() method checks for collisions with zombies and reduces their health accordingly.
- The advance() method moves the bullet forward on the screen.
- **PeaBullet Class**:
- This class extends Bullet and represents a basic pea bullet shot by plants.
- It sets the bullet's image icon and inherits the collision detection method from the base class.
- **SnowPeaBullet Class**:
- This class also extends Bullet and represents a snow pea bullet, which not only damages zombies but also slows them down.
- It overrides the collision detection method to include the slowing effect on zombies in addition to damaging them.

- Package Game:
  - ·BGM class:
  - The BGM class extends Thread and continuously plays background music in a loop. It defines constants for the music file path and buffer size. The overridden run() method repeatedly calls a private playMusic() method. This method uses AudioSystem.getAudioInputStream() to read the music file, retrieves the audio format, and opens a SourceDataLine. It then reads audio data into a buffer and writes it to the SourceDataLine, ensuring all data is played by draining the line. The code handles exceptions for unsupported audio files, IO issues, and unavailable lines. This setup allows for continuous background music in a Java application.
  - End class:
  - The End class extends JPanel and represents a game end screen. It sets the panel size to 1400x637 pixels and uses manual component positioning. The class loads an "endImage" from "ZombiesWon.jpg" and plays a "gameOver.wav" sound. A mouse listener triggers the selectModel() method of the GameWindow class when the panel is clicked. The paintComponent() method draws the endImage at (0, 0). This setup creates an end screen with an image and sound effect, allowing the user to initiate an action, such as restarting the game or returning to a menu, by clicking the panel.
  - GamePlay class:
  - The GamePlay class manages the game's states and components. Key attributes include imageIcon, preImage, grasses grid, shovel, sunProducer, sunLabel, zombieController, bgm, and closePreImageTimer. The constructor sets panel size and visibility, initializing modes like the main menu, gameplay, end game, or game win. Important methods include setupPlayMode() and various initializations, plus methods for

checking rows for zombies, removing the preparation image, and accessing components. Overall, it coordinates game setup, management, and state transitions.

- GameWindow class:
  - The GameWindow class manages the main game window and transitions between game states. It features a gameWindow attribute (the main JFrame) and an Enum GameState with states (MENU, PLAY, END, WIN) linked to specific modes and dimensions. The main method sets the initial state to MENU. The setGameState method updates the game state, creating a new JFrame and GamePlay object, and configuring the window. Methods like selectModel, startPlay, gameEnd, and gameWin handle state changes. Overall, GameWindow controls the graphical interface, state management, and interactions with the GamePlay panel.
- Grass class:
  - The Grass class represents grass tiles where plants can be planted. Key attributes include coordinates (X, Y), dimensions (WIDTH, HEIGHT), an isAssigned flag, the assigned plant (assignedPlant), timers for checking plant state and row zombies, and a reference to the GamePlay object. The constructor sets the tile's position and starts a timer to monitor the plant's state. Key methods are checkState() for monitoring the plant, removeAssignedPlant() for removing a plant, setAssignedPlant() for assigning a plant and updating its state, loseLife() for decreasing plant life, and setIsShooting() for setting plant shooting based on nearby zombies. Getter methods provide tile position and assignment status. Overall, Grass manages tile state, plant occupancy, and related behaviors.
- · Menu class:
  - The Menu class represents the game's main menu panel. It sets the panel's size, visibility, and layout, and loads images for the background and start button. It includes attributes for the panel and button dimensions and positions. The class adds a mouse listener to handle clicks. Key methods include paintComponent(Graphics g) to draw the menu and button, and handleMouseClick(MouseEvent e) to start the game if the start button is clicked. Overall, the Menu class handles the visual display and interactions for starting the game from the main menu.
- Shovel class:
  - The Shovel class facilitates plant removal from the game grid. It includes attributes like layeredPane, initial coordinates (X, Y), mouse coordinates (x0, y0) for dragging, and dimensions (WIDTH, HEIGHT) of the shovel. It registers mouse listeners for handling events. The mouseReleased() method removes plants at the release location, while mouseDragged() moves the shovel. Overall, it enables players to interactively remove plants from the game grid using mouse actions.
- Sound class:
  - The Sound class manages audio playback in the game. It has attributes for controlling sound cessation (isStop), the audio file's URL (url), and a flag for looping (loopNeed). The constructor initializes these attributes. The run() method continuously plays the sound if looping is needed, otherwise, it plays it once. playSound() method handles the

audio playback process. Overall, Sound provides a simple and effective solution for handling sound effects in the game.

- Sun class:The Sun class manages sun objects in the game. Key attributes include coordinates (X, Y), descent speed (ySpeed), and a reference to the JLayeredPane. The constructor initializes the sun and starts a timer for descent, adding a mouse listener for interactions. Methods include drop() for descent and mouseClicked() for removing the sun, updating sun count, repainting, and playing a sound effect. Overall, the Sun handles sun behavior and interaction in response to user input and game mechanics.

- SunLabel class: The SunLabel class serves as the label displaying the current sun count in the game. It contains an attribute 'total' representing the total number of suns available. Upon instantiation, the constructor initializes the sun count to 100 and configures the label's appearance and position, adding it to the layered pane. The class provides methods for retrieving the current sun count and dynamically updating it during gameplay. Overall, SunLabel enhances player awareness by visually representing the available suns and facilitating interaction within the game environment.

- SunProducer class:
  - The SunProducer class manages the consistent production of suns during gameplay. It utilizes a timer to trigger sun production at regular intervals and generates random X coordinates for sun placement. The produceSun() method creates new Sun objects at random X coordinates within the playable area. Overall, SunProducer ensures a steady supply of suns, vital for player progression in the game.
- Win class:
  - The Win class represents the victory screen in the game. It sets the panel size, plays a victory sound effect, and adds a mouse listener for clicks. The paintComponent(Graphics g) method draws the background image. When clicked, mouseClicked(MouseEvent e) triggers the GameWindow class's selectModel() method, allowing players to return to the main menu. Overall, Win provides a concise and effective way for players to navigate back to the main menu after winning the game.
- Yard class:
  - The Yard class visually represents the game grid, setting the stage for plant and zombie interactions. It sets the panel size, ensures visibility, and uses a null layout manager. The paintComponent(Graphics g) method draws the background images of the yard. Overall, Yard creates the game environment with minimalistic setup, focusing on rendering background images for gameplay.
- ZombieController class:
  - The ZombieController class manages the generation, behavior, and resolution of zombies in the game. It handles the timing and creation of zombies, monitors their states,

orchestrates movements and attacks, and oversees special events like wave announcements and win conditions. This class is crucial for shaping the game's pace and dynamics, ensuring an engaging and immersive experience for players.

- Package Plant:
    - · The Plant class serves as the base class for various types of plants in the game. It encapsulates common functionality required by all plant types. Key attributes include WIDTH and HEIGHT for dimensions, sunUse for the amount of sunlight required, life for remaining life, gamePlay for reference to the game instance, redrawTimer for repainting, plantImage for the plant's image, and px and py for position. Methods include Plant(width, height, sunUse, gamePlay) for initializing attributes, paintComponent(g) for rendering the plant, getGamePlay() for returning the game instance, and getSunUse() for returning the sunlight requirement.
    - · The PotatoMine class extends Plant and represents a plant that explodes when triggered by nearby zombies. It includes additional attributes such as stateChangeTimer and bombTimer for state changes and explosions, and state for the potato mine's current state. Methods include PotatoMine(gamePlay) for initialization and timer start, stateChange() for transitioning to a ready state, timerStop() for stopping timers, and bomb() for executing the explosion.
    - · The Sunflower class, also extending Plant, generates sunlight periodically. It includes a sunProduceTimer for this functionality. Methods include Sunflower(gamePlay) for initialization and timer start, and produceSun() for generating sunlight.
    - · The Peashooter class, another Plant extension, shoots peas at zombies. It includes shootTimer for shooting intervals and isShooting to indicate shooting status. Methods include Peashooter(gamePlay) for initialization and timer start, and shoot() for shooting bullets.
    - · The SnowPeaShooter class, a specialized version of Peashooter, shoots frozen peas that slow down zombies. It overrides the shoot() method to instantiate SnowPeaBullet objects. Methods include SnowPeaShooter(gamePlay) for initialization and setting the plant image, and shoot() for shooting frozen bullets.
    - · The WallNut class extends Plant and acts as a barrier against zombies. It includes stateChangeTimer for state changes and counter for tracking these changes. Methods include WallNut(gamePlay) for initialization and timer start, and stateChange() for changing the state when damaged.
    - · The Chomper class extends Plant and devours zombies in its range. It includes biteTimer and resetTimer for biting and resetting actions, and state for the chomper's current state. Methods include Chomper(gamePlay)

for initialization and timer start, bite() for executing the biting action, reset() for resetting after biting, and timerStop() for stopping timers.

- · The CherryBomb class, also extending Plant, explodes to eliminate nearby zombies. It includes bombTimer and counterTimer for explosion timing and animation frame counting, and counter for tracking animation frames. Methods include CherryBomb(gamePlay) for initialization and timer start, paintComponent(g) for painting the cherry bomb animation, count() for incrementing the animation frame counter, and bomb() for executing the explosion.
- · These classes collectively represent various types of plants within the game, each with unique behavior and functionality. They interact with the game environment, handle user interactions, and contribute to the overall gameplay experience.

- Package Plant Card:
  - **Role:** The PlantCard class is the base class managing the behavior and properties of plant cards in the application. It handles mouse events and manages the graphical display.

Attributes:

- layeredPane: An instance of JLayeredPane used to manage components in a specific layer.
- X: An integer representing the x-coordinate of the card.
- Y: An integer representing the y-coordinate of the card.
- x0: An integer representing the initial x-coordinate of the card.
- y0: An integer representing the initial y-coordinate of the card.
- dragCounter: An integer used to track drag events.
- PlantCardImage: An ImageIcon instance representing the image of the card.
- WIDTH: An integer representing the width of the card.
- HEIGHT: An integer representing the height of the card.

Methods:

- void mouseClicked(MouseEvent): Handles mouse click events.
- void mousePressed(MouseEvent): Handles mouse press events.
- void mouseReleased(MouseEvent): Handles mouse release events.
- void mouseEntered(MouseEvent): Handles mouse enter events.
- void mouseExited(MouseEvent): Handles mouse exit events.
- void mouseDragged(MouseEvent): Handles mouse drag events.
- void mouseMoved(MouseEvent): Handles mouse move events.
  - Subclasses of PlantCard
- These classes inherit from PlantCard and each has the void mouseReleased(MouseEvent) method.

- CherryBombCard

Role: Represents the Cherry Bomb plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Cherry Bomb card.

- PeaShooterCard

Role: Represents the Pea Shooter plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Pea Shooter card.

- PotatoMineCard

Role: Represents the Potato Mine plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Potato Mine card.

- SnowPeaShooterCard

Role: Represents the Snow Pea Shooter plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Snow Pea Shooter card.

- SunflowerCard

Role: Represents the Sunflower plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Sunflower card.

- ChomperCard

Role: Represents the Chomper plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Chomper card.
  - WallNutCard

Role: Represents the Wall-Nut plant card in the game.

Attributes and Methods:

- Inherits all attributes from PlantCard.
- void mouseReleased(MouseEvent): Handles mouse release events specific to the Wall-Nut card.
- Inheritance and Interfaces
- PlantCard inherits from JLabel, meaning it has all the attributes and methods of JLabel.
- PlantCard implements the MouseListener and MouseMotionListener interfaces, meaning it must provide methods to handle mouse events as described above.

- Package Zombie:
  - Zombie class:
    - The "Zombie" class in the "Zombie" package extends JPanel to manage zombie entities in a graphical environment. It encapsulates attributes like width, height, position, speed, and life, along with a State enumeration for the zombie's state. Methods handle painting, advancing, resetting speed, and modifying state. It provides functionality to create, display, and manage zombies, enabling movement, attacks, and state changes.
  - Normal Zombie class:
    - "NormalZombie" extends the "Zombie" class to handle a specific type of zombie in the game. It inherits attributes and methods, initializing instances with specific position, width, height, and layer placement. The class sets the initial life to 10 and loads images for various states like advance, attack, frozen advance, frozen attack, dead, and head. This specialization enables the management and visual representation of normal zombies within the game environment.
  - Flag Zombie class:
    - "FlagZombie" is another subclass of "Zombie" tailored for a special type of zombie in the game. It inherits basic zombie attributes but adjusts properties like initial position, vertical displacement, and size uniquely for flag zombies. It initializes with a life of 10 and loads specific images for states like advance, attack, frozen advance, frozen attack, dead, and head. This class focuses on managing and visually distinguishing flag zombies within the game environment.

This zombies only appear when started the waves and only 1 zombie in the waves has a flag.

- ConeheadZombie class:
  - The "ConeheadZombie" class extends "Zombie" to manage conehead zombies in the game. It inherits the basic properties of zombies but adjusts properties such as initial position, vertical displacement, and size specifically for conehead zombies. This subclass creates cone-headed zombies with a lifespan of 20 and loads unique textures for states such as advance, attack, frozen advance, frozen attack, dead, and headed. It focuses on managing and displaying conehead zombies with appropriate attributes and images in the game environment.
- BucketheadZombie class:
  - The "BucketheadZombie" class extends "Zombie" to manage bucket head zombies in the game. It inherits the basic properties of the zombie while adjusting specific properties such as initial position and size specifically for bucket head zombies. This subclass instantiates bucket-head zombies with a lifespan of 40 and loads adjusted images for states like advance, attack, frozen advance, frozen attack, dead, and headed from resources. It focuses on handling and displaying bucket-head zombies with specialized attributes and visuals in the game environment.

III. Game Design:

a) Background:

The game background includes a lawn where the battle between plants and zombies takes place. The design aims to create an engaging and visually appealing environment for players. These are the arts manipulating in our games:

Figure 3a.1: Menu background

b) Lawn Design:

- The lawn is the primary battlefield, divided into a grid format (typically 5 rows by 9 columns) where plants and zombies interact.
- The grid layout allows for the strategic placement of plants.
- The lawn background includes visual elements such as grass, pathways, and a backdrop of the player's house which the zombies aim to reach.

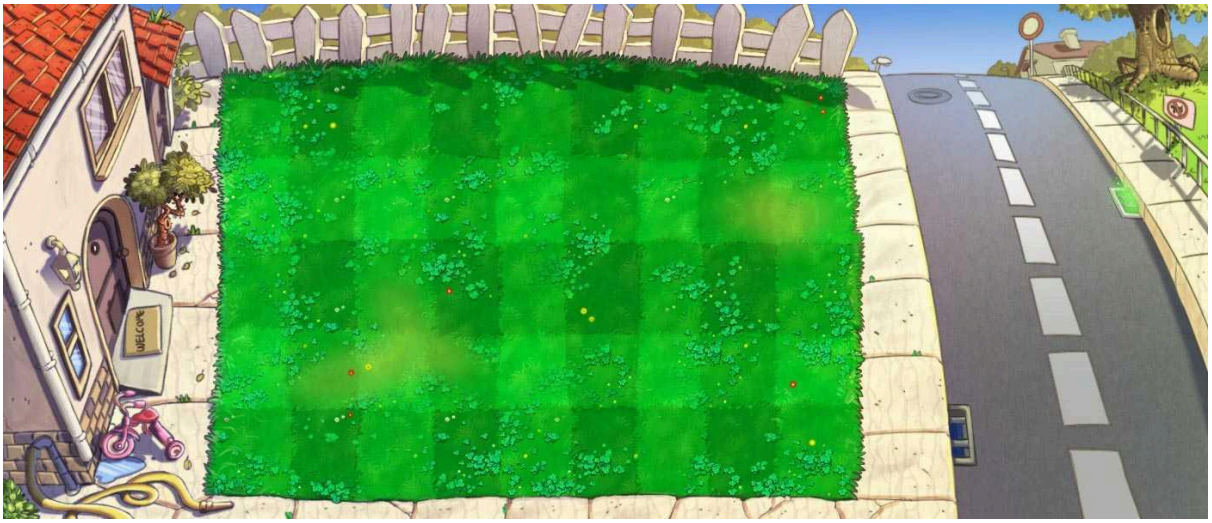


Figure 3a.2: column background

c) Visual Appeal:

- High-quality, cartoon-style graphics are used to make the environment vibrant and engaging.
- Attention to detail, such as shadows, animations of plants and zombies, and background elements, enhances the overall visual experience.

d) Frame:

The game frame is the main window that encapsulates all game elements. It serves as the interface through which players interact with the game.
- Structure:

The frame includes the lawn, plant selection panel, sun point display, and control buttons. It is designed to be user-friendly and intuitive, ensuring players can easily navigate and interact with the game elements.

- Components:

- Lawn Grid: The central area where the plants and zombies are placed and interact.
- Plant Selection Panel: Displays available plants and their sun point costs.
- Sun Point Display: Shows the current amount of sun points available to the player.
- Control Buttons: Includes buttons for starting the game, pausing, restarting, and accessing settings.

Figure 3a.3: Menu background

Figure 3a.4: Menu background

e) Plants:

Plants are the primary defense mechanism in the game. Each plant has unique attributes and abilities that contribute to the player's strategy.

- Attributes:
    - Health: The amount of damage a plant can take before being destroyed.
    - Cost: The number of sun points required to plant it.
    - Attack Type: The method by which the plant attacks (e.g., shooting projectiles, producing sun points).
    - Special Abilities: Unique abilities that some plants possess, such as slowing down zombies or causing area damage.
- Examples:
    - Peashooter: Shoots peas at zombies. Cost: 100 sun points.
    - Sunflower: Produces additional sun points. Cost: 50 sun points.
    - Wall-nut: Acts as a barrier to delay zombies. Cost: 50 sun points.
- Plant:

Figure 3a.5: SnowPea Plant

Figure 3a.6: WallNut Plant

Figure 3a.7: Peashooter Plant

Figure 3a.8: Chomper Plant

Figure 3a.9: PotatoMine Plant

Figure 3a.10: SunFlower Plant

f) Zombies:

Zombies are the adversaries that attempt to reach the player's house. Different types of zombies have varying attributes that affect how they interact with the plants and progress across the lawn.

- Attributes:
    - Health: The amount of damage a zombie can take before being defeated.
    - Speed: The rate at which a zombie moves across the lawn.
    - Damage: The amount of damage a zombie deals to plants when attacking.
    - Special Abilities: unique zombies possess unique abilities, such as wearing protective gear or summoning other zombies.
- Examples:
    - Normal Zombie: standard zombie with moderate health and speed.
    - Conehead Zombie: Wears a traffic cone for extra protection, increasing its health.
    - Buckethead Zombie: Wears a bucket for even more protection, making it harder to defeat.
- Zombie:

Figure 3a.11: Conehead Zombie Image

Figure 3a.12: Flag Zombie Image

Figure 3a.13: Buckethead Zombie Image

Figure 3a.14: Zombie Normal Image

g) Button:

Buttons are essential for player interaction and control within the game. They provide functionality for starting the game, selecting plants, pausing, and restarting the game.

- Types of buttons:
    - Start Button: Initiates the game. Typically, this leads to the first wave of zombies.
- Design Considerations:
    - Buttons should be clearly labeled and easily accessible.
    - Visual feedback (such as highlighting or animation) when buttons are pressed enhances the user experience.

h) GameState:

The GameState class is a crucial component that manages the current state of the game. It keeps track of active plants and zombies, sun points, and overall game progress.

- Responsibilities:
    - Tracking Entities: Manages lists of active plants and zombies on the lawn.
    - Sun Points: Keeps track of the player's current sun points, updating the display as points are collected or spent.

IV. Result:

a) Achievements:

- **Java Proficiency:**
  - **Enhanced Java Skills**: The project significantly improved the team's Java programming skills, especially in areas like object-oriented programming (OOP), multi-threading, and event-driven programming. Team members are now more confident in using Java for complex software development tasks.
- **OOP Principles Applied**:
  - **Effective Use of OOP**: The game development process reinforced core OOP principles such as inheritance, polymorphism, encapsulation, and abstraction. Each game element (plants, zombies, and game states) was modeled as a class, showcasing a strong grasp of OOP design patterns and principles.
- **Game Development Experience**:
  - **Comprehensive Game Development**: The team successfully designed and implemented a fully functional game. This includes creating a robust game engine, handling real-time user inputs, managing game states, and ensuring smooth gameplay. The project covers all major aspects of game development from initial concept to final deployment.
- **Collaboration and Project Management**:
  - **Effective Team Collaboration**: Utilizing tools like Git/GitHub for version control and task management software, the team efficiently collaborated and managed the project. This experience improved their skills in working on large-scale projects, ensuring code quality, and integrating different modules seamlessly.
- **Feature Implementation and Extension**:
  - **Scalable Architecture**: The game was designed with scalability in mind, allowing for easy addition of new features and enhancements. For example, new types of plants and zombies can be added without major changes to the existing codebase, demonstrating a flexible and modular design approach.
- **Graphic and Audio Integration**:
  - **Multimedia Skills**: The project involved creating and integrating various multimedia elements, including sprites, background images, and audio effects. The team effectively used tools like Adobe Photoshop, Pixel Studio, and PineTools to enhance the visual and auditory appeal of the game.
- **User Experience and Interface Design**:

- **Intuitive UI/UX**: The game features a user-friendly interface with clear navigation and interactive elements like buttons and menus. The design prioritizes user engagement and ease of use, providing a smooth and enjoyable gaming experience.

b) Goals Summary:

- **Educational and Skill Development**:
  - **Learn and Practice Java**: Achieved through hands-on development and problem-solving during the project.
  - **Improve OOP Principles**: Demonstrated by the structured and object-oriented design of game components.
- **Continuous Improvement and Management**:
  - **Iterative Development**: Implemented by regularly updating and refining the game based on testing and feedback.
  - **Feature Enhancement**: Achieved by adding new plant and zombie types, ensuring the game remains engaging and challenging.
- **Teamwork and Large-scale Project Execution**:
  - **Collaborative Development**: Achieved through effective use of version control, task management, and regular team meetings.
  - **Project Setup and Execution**: Successfully managed a complex project with multiple interconnected parts, ensuring timely delivery and quality output.

c) In-game menu interface:

Figure 1: Start

Figure 2: ingame

Figure 3: ingame

V. Conclusion and Future Works:

1. Conclusion:

During the development of our Plant vs. Zombies-inspired game, our team learned a lot about the four main ideas of object-oriented programming (OOP): inheritance, polymorphism, abstraction, and encapsulation. Understanding these ideas better helped us use OOP principles in making the game, allowing us to add new features beyond the first version. Our game now includes all four important OOP principles and uses a design pattern we learned in our course, showing our commitment to these concepts. Our team's dedication to these values highlights our commitment to making this project successful.

2. Future Work:

Our game will keep getting updates in the future. One main focus will be adding new level with new maps that have new zombies and plant, making the game more surprising and varied. Another exciting addition will be creating various plants with unique abilities, giving players more choices. These future improvements aim to make the game more fun and engaging for players

3. Acknowledgments:

We want to thank our instructor and everyone who helped us achieve the project's goals:

Dr. Tran Thanh Tung
MSc. Nguyen Trung Nghia

VI. REFERENCES:

For Graphics of this Game: FandomPLZ
For Sample Code: Java Tower Defence Tutorial
For Game Design: PLZ from the other