VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



**MONOPOLY GAME REPORT**

|  |  |  |
| --- | --- | --- |
| Name | ID | Contribution |
| Nguyễn Dương Tấn Đạt | ITDSIU20096 | % |
| Nguyễn Minh Quân | ITDSIU20094 | % |
| Phan Thế Thiện | ITDSIU20084 | % |
| Nguyễn Phạm Gia Linh | ITDSIU20068 | % |
| Phan Đăng Hưng | ITDSIU20119 | % |

Ho Chi Minh City, Vietnam

2022-2023

# **1. Table of Contents**

[1. Topic Motivation 2](#_Toc134213381)

[2. Table of Contents 3](#_Toc134213382)

[3 Game Rule and Game Loop 3](#_Toc134213383)

[3.1 Game Rule 4](#_Toc134213384)

[3.2 Game Loop 5](#_Toc134213385)

[Fig 3.2.1 Game loop 5](#_Toc134213386)

[4 System Design 5](#_Toc134213387)

[4.1 List of Classes and their Responsibilities: 5](#_Toc134213388)

[4.2 Class Diagram 8](#_Toc134213389)

[Diagram 4.2.1 8](#_Toc134213390)

[4.2.1 Player and Bot Class Diagram 9](#_Toc134213391)

[Diagram 4.2.2 9](#_Toc134213392)

[4.2.1 Collidable Class Diagram 10](#_Toc134213393)

[4.3 Game feature 11](#_Toc134213394)

[4.3.1 Dice  
4.3.2 Player Figure 11](#_Toc134213395)

[4.3.2 Chance card  
4.3.3 Building  
4.3.4 Roll Button 12](#_Toc134213396)

[5. CONCLUSION 16](#_Toc134213400)

[5.1 What can be improved next time? 17](#_Toc134213401)

# **2. Topic Motivation**

* + As a team working on a game project for the first time, we have chosen to create a Monopoly game. Monopoly has become a part of international popular culture, having been licensed locally in more than 103 countries and printed in more than 37 languages. As of 2015, it was estimated that the game had sold 275 million copies worldwide.
  + Monopoly is derived from The Landlord's Game, created by Lizzie Magie in the United States in 1903 as a way to demonstrate that an economy that rewards individuals is better than one where monopolies hold all the wealth and to promote the economic theories of Henry George—in particular, his ideas about taxation.(according to Wikipedia)
  + We believe that this project presents an excellent opportunity to explore and expand upon the gameplay elements. By recreating a Monopoly game, we hope to meet the requirements of the university course and create a product that is enjoyable and challenging for player.

# **3. Game Rule and Game Loop**

## 3.1 Game Rule

* Monopoly, real-estate board game for two to four players, in which the player’s goal is to remain financially solvent while forcing opponents into bankruptcy by buying and developing pieces of property.
* Each side of the square board is divided into 10 small rectangles representing specific game function such as prison, bus station and various other places.
* At the start of the game, each player is given a fixed amount of play money; the players then move around the board according to the throw of a pair of dice. Any player who lands on an unowned property may buy it, but, if he or she lands on a property owned by another player, rent must be paid to that player. Certain nonproperty squares require the player landing on them to draw a card that may be favourable or unfavourable. If a player [acquires](https://www.britannica.com/dictionary/acquires) a monopoly—that is, all of a particular group of properties—that player may purchase improvements for those properties; improvements add substantially to a property’s rental fee. A player continues to travel around the board until he or she is bankrupt. Bankruptcy results in elimination from the game. The last player remaining on the board is the winner.
* There are some special case make player win and end the game even though he or she is not the last player

## 3.2 Special Win Game Rule

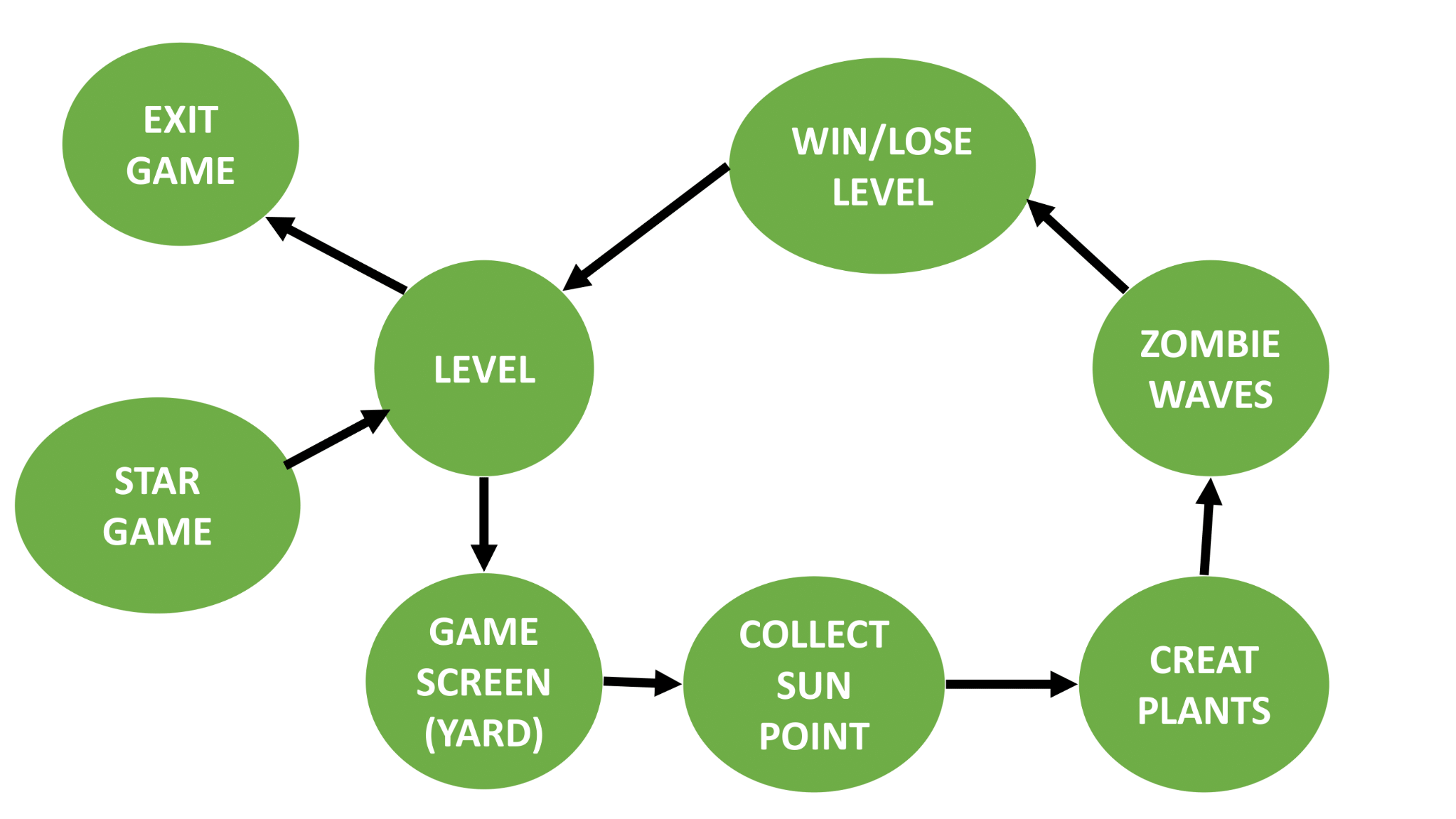
* Player will win the game if:

- Player has buildings which are located on one of four square sides

- Player has four beaches.

- Player has a highest property when the number of turns equals the chosen limitation of turn

## 3.3 Game Loop



#### Fig 3.2.1 game loop

# 4 System Design

## 4.1 List of Classes and their Responsibilities:

## 

|  |  |  |
| --- | --- | --- |
| Number | Class Name | Responsibility |
| 1 | Button.java | represents a button element in a graphical user interface and has the ability to interact with the game. |
| 2 | Card.java | a subclass of the JPanel class and implements the MouseListener and GameObject interfaces. It represents a card element in a graphical user interface and has the ability to handle mouse events and interact with the game. |
| 3 | PlantCard.java | plant a plant in a game by clicking on the card. It contains a PlantType enum representing different plant types and an action event triggered when the card is clicked. It also overrides the paintComponent method to draw an image of the plant and the mouseReleased method to handle the card click. |
| 4 | ShovelCard.java | a imagethat can be used to remove a plant in a game. It overrides the paintComponent method to draw an image representing the shovel and the mouseReleased method to handle a user clicking the card. |
| 5 | Collider.java | The Collider class represents a GUI element that can handle mouse events and has the ability to assign and remove a Plant object. It has methods for checking if a given x coordinate is inside its bounds and for setting an action listener. |
| 6 | GameObject.java | GameObject is an interface that specifies the methods that a class must implement. |
| 7 | GamePanel.java | GamePanel is a JLayeredPane that manages the game logic, graphics, and mouse events, including colliders, lanes of zombies and peas, and active suns. It also includes inner classes for plant and shovel actions. |
| 8 | GameWindow.java | GameWindow is a JFrame that displays the game and handles user input through PlantCards and ShovelCard buttons to change the active planting brush and active shovel, respectively. It also displays a sun label and has a GamePanel as a layered pane when the game begins. |
| 9 | Progress.java | The Progress class manages the progress made in a game level and displays a message when a level is completed |
| 10 | ProgressBar.java | calculate the player progress and show the menu when the player win tbe level |
| 11 | ImageContainer.java | "ImageContainer is a class that stores and retrieves images used in the game. It has methods to load images from file paths and getters for each image used in the game." |
| 12 | LevelData.java | storing and reading level data from a file. It has a static Lvl field which stores the current level, and two static 2D arrays that store data about the level. There is also a constructor and a write method for reading and writing the level data to a file. |
| 13 | Menu.java | a class representing the menu screen in a game. It displays a menu image and has a button that, when clicked, starts the game by calling the begin() method of the GameWindow class. |
| 14 | Defender.java | Defender is a type of Plant that has extra health and is a child of Plant.java. |
| 15 | Shooter.java | represents a plant that can shoot in a game. It has a Timer field, shootTimer, which is used to control when the plant shoots |
| 16 | ShovelButton.java | a subclass of the Button class and represents a button that can be used to remove plants in a game |
| 17 | Yard.java | Representing the background of the game |
| 18 | LawnMower.java | The LawnMower class is a game object that represents a lawn mower. It moves along its lane and removes any zombies it intersects with. Its behavior is updated periodically through the check and mowing methods. |
| 19 | Character.java | represents a character element in the game and has attributes for its position (x and y coordinates) and health. It also has getter and setter methods for these attributes. |
| 20 | Zombie.java | The Zombie class represents a zombie in a game. It has a speed and attack value and is associated with a lane in the game. The class has a method called eating which handles the movement and attack of the zombie. It also has a method called slow which slows down the movement of the zombie |
| 21 | Zomboni.java | The Zomboni class is a type of Zombie that has extreme high attack damge and can remove most plant types |
| 22 | ConeHeadZombie.java | ConeHeadZombie is a type of Zombie that has increased attack and health, and moves at a normal speed, and is a subclass of Zombie. |
| 23 | NormalZombie.java | A NormalZombie is a type of Zombie in the game. |
| 24 | Plant.java | The Plant class is a superclass for characters in the game with a method to remove itself and a method to stop. It also has a reference to the game panel in which it exists. |
| 25 | PlantButton.java | The PlantButton class is a subclass of Button and has an enumerated type PlantType that contains the different types of plants that can be planted in a game. |
| 26 | FreezePeashooter.java | FreezePeashooter is a shooter plant that shoots freezing peas and is a child of Shooter.java |
| 27 | GatlingPea.java | GatlingPea is a type of Pea that deals 250 damage and is used to shoot at Zombies in the game. |
| 28 | GatlingPeashooter.java | GatlingPeashooter is a type of Shooter that shoots GatlingPea at a rate of 2000 milliseconds in a lane and has 3 GatlingPea in a line. It is a child of Shooter class. |
| 29 | GreenPea.java | A type of pea that deals 300 damage. |
| 30 | Pea.java | Pea class has a method called "shoot" which moves the pea and checks for collisions with zombies. If the pea collides with a zombie, the zombie's health is reduced by the pea's damage value and the pea is removed from the game panel. If the zombie's health is below 0, it is removed from the game panel and the game's progress is updated. |
| 31 | Peashooter.java | Peashooter is a type of plant in a game that shoots GreenPeas at zombies to damage them. The Peashooter shoots every 2 seconds if there are zombies in its lane. |
| 32 | Sun.java | The Sun class is a subclass of JPanel and represents a sun in a game. It has a product method which moves the sun downward until it reaches the endY position. The sun also has a destruct timer which decreases every time the sun is not collected by the player. |
| 33 | Sunflower.java | The Sunflower class extends the EnergyProduct class and represents a plant that produces suns in a game. It has a timer that produces a new sun every 15 seconds and adds it to the game panel. |
| 34 | SunScore.java | The SunScore class is a simple class that represents the player's current score in sun points in a game. It has a sunScore field that stores the current sun score, a sunScoreboard JLabel that displays the current sun score to the player, and a setSunScore method that updates the sunScore field |
| 35 | WallNut.java | a type of plant in a game that is used to defend against zombies. It extends the Defender class, which is a type of plant that can defend against zombies. The WallNut class has an extra health of 1800 |

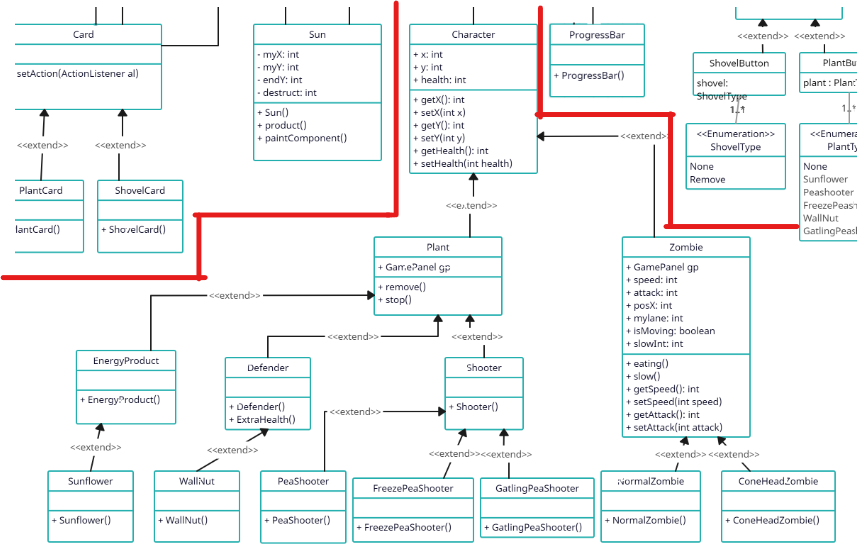
## 

## 4.2 Class Diagram

#### Diagram 4.2.1

The diagram above shows the complete relationship of all classes in our game.

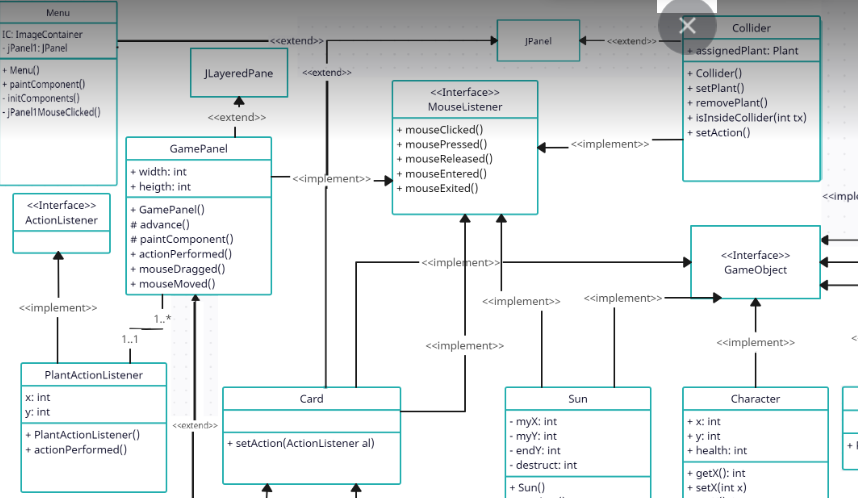
##### 4.2.1 Plant and Zombie character



#### Diagram 4.2.2

First, let's have a look at this part of the diagram. The character class and everything inherited from this class. Both plant and Zombie are from the superclass Character with an x and y coordinate and a health value.

##### 4.2.1 Collidable Class Diagram



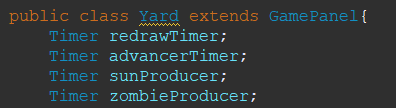
Next is the collision when a pea from a peashooter hit a zombie and when a zombie reaches any of the plants:

It is used to detect collisions or interactions between objects in the game. It is used to determine when a plant or zombie comes into contact with another plant or zombie and can be used to trigger actions or events based on these interactions. In this game, colliders are used to detect when a zombie is attacking a plant and can be used to remove a plant from the game when its health is reduced to zero. Colliders are also used to prevent plants from being placed on top of each other, and to determine when a zombie has reached the end of the lane and the player has lost the game.

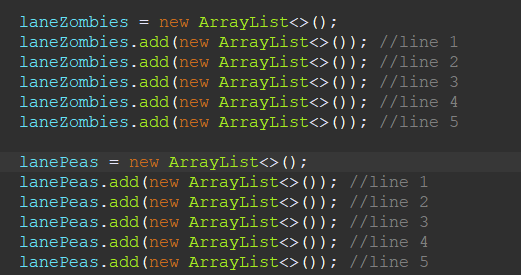
In conclusion, the class diagram for the game system effectively captures the main components and relationships of the system

## 4.3 Game feature

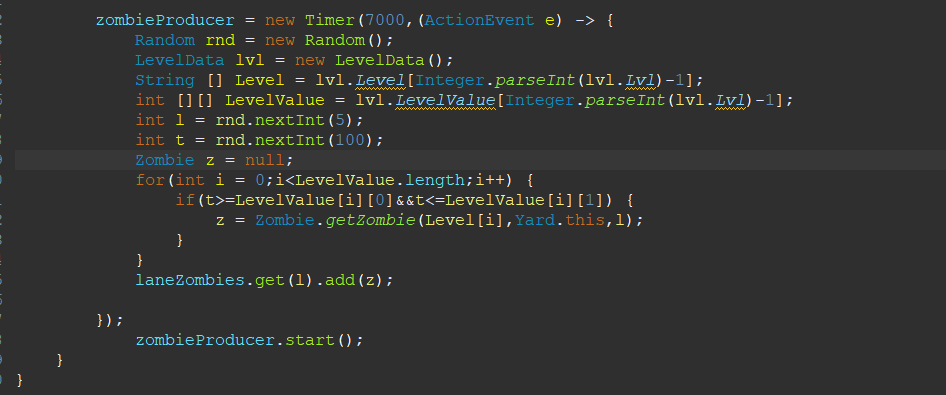
### 4.3.1 Zombie generator



This section declares four Timer objects that will be used to schedule events in the game. redrawTimer will be used to redraw the game panel at regular intervals, advancerTimer will be used to advance the game state, sunProducer will be used to produce suns at regular intervals, and zombieProducer will be used to produce zombies at regular intervals.



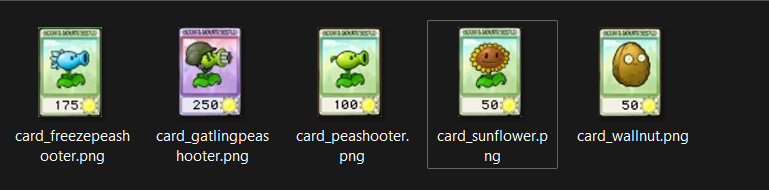
This section creates two ArrayList objects, laneZombies and lanePeas. These lists will be used to store the zombies and peas in the game, respectively. Each list has five sublists, one for each lane in the game.



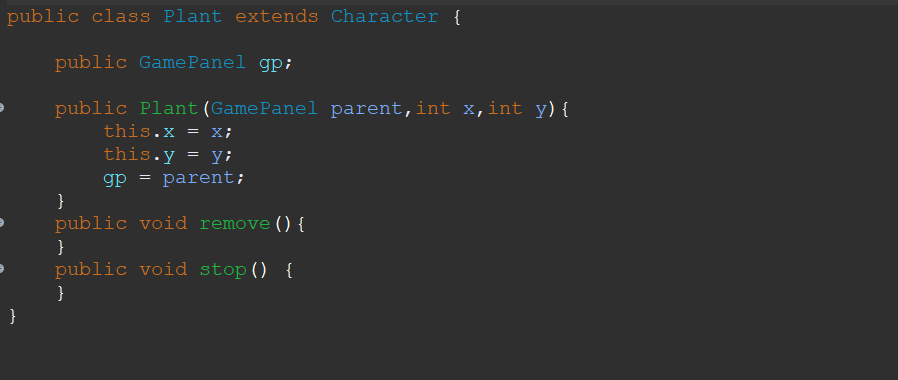
The zombieProducer timer is responsible for generating new zombies in the game at intervals of 7000 milliseconds (7 seconds). When the timer is triggered, it performs the following actions: initializes a Random object(such as which

types of zombies should be generated and at what probabilities); retrieves the current level's zombie data; initializes a Zombie object.; loops through the LevelValue array and uses the probability value to determine which type of zombie to create.

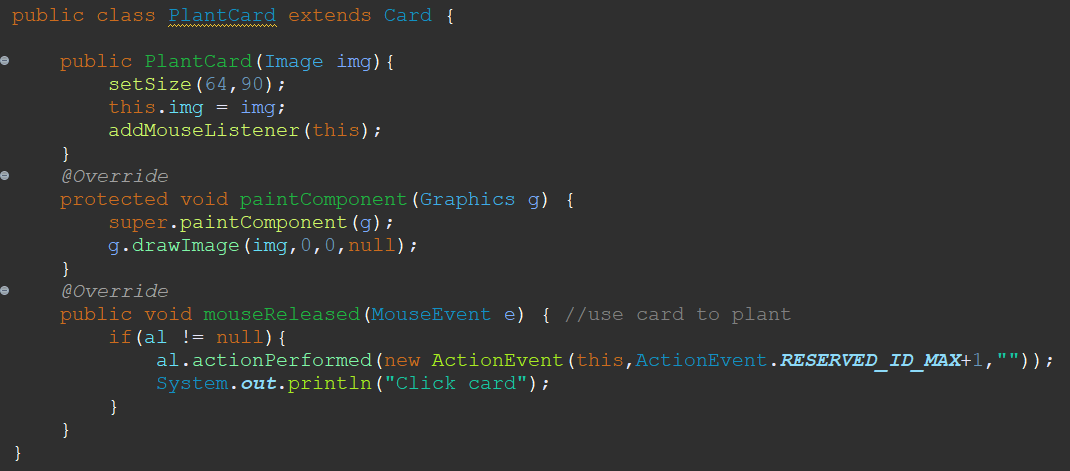
### 4.3.2 Plant and plant’s card

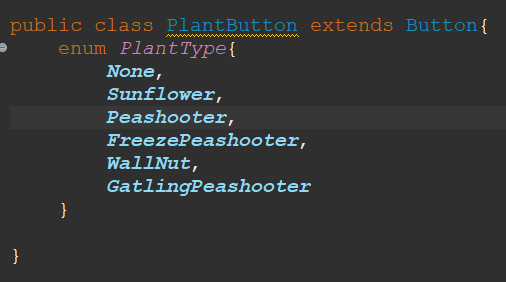


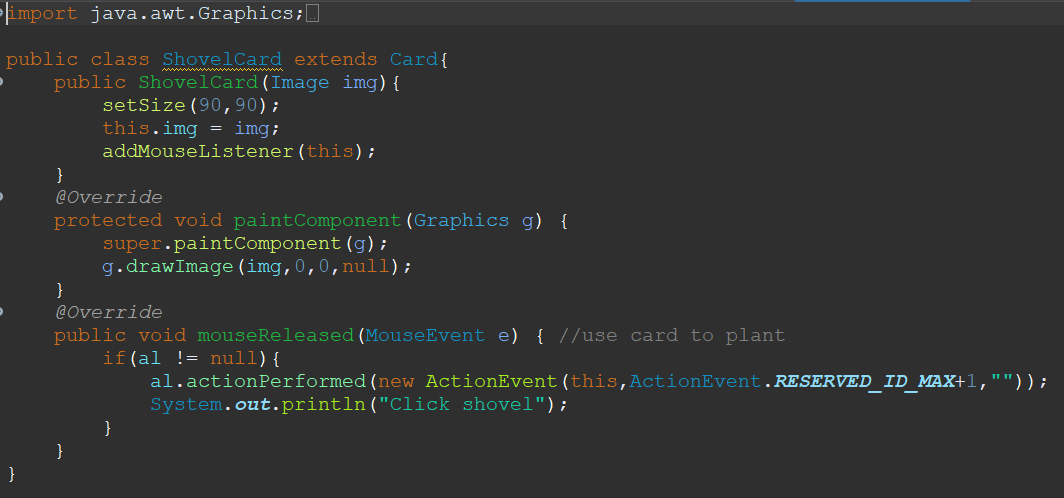
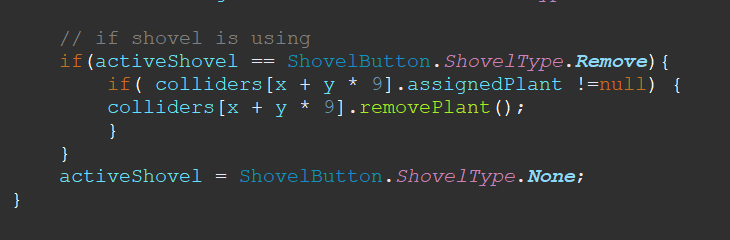
All plants and their corresponding cards: each card represent a type of plant that can be planted in 5 of the line in the game. The main Plant class have some attribute of a plant in game are position(x,y), health and 2 method remove() in case they were attack by zombies or shoveling by user, stop() in this case it use for EnergyProduct and Shooter class.



The class PlantCard extends Card class which implements MouseListener, base on this, PlantCard can receive mouse click from user. Along with that we also create a PlantButton class have PlantType enum( None, Sunflower, PeaShooter, FreezePeaShooter, GatlingPeaShooter, WallNut) to choose type of plants refer to planting.







### 

# 5. CONCLUSION

In this game, we have implemented various features including a system for generating different type of map, dices, chance cards, sound effect, images and figures. Each chance has its own associated card system. We have also implemented an object-oriented approach, with player and bot able to interact the gameplay and various effects being applied to the gameplay according the event. Animations have also been added to all of the game's characters within the user interface.

This game is solely made for the better understanding of the OOP and we try as hard as we could to apply the SOLID principle, not saying that our game are entirely optimal and followed the SOLID principle but the game was made with that intention.

## 5.1 What can be improved next time?

* Do more research before choosing a game project, make a list of advantages and disadvantages.
* Carefully choose teammates who can help and support us in order to release better project.
* Find a high-quality source of images for the game visual style for avoiding low, pixel image lead to eyes irritation
* Add chance types: Adding more variety to the game can make it more interesting and challenging for players.
* Improve game interface: Redesign the game interface such as animation, colors, sound effects, themes to catch the player’s eyes.
* More real player(s): This project only allows one player playing with bot(s) so it must allow more than one player can play at the same time.