Object-Oriented Programming report

A simple game written in Java applying OOP principles

## **Members**

Tran Huu Phuc - ITDSIU19013

Nguyen Minh Trang - ITDSIU19020

Phan Vo Phuong Tung - ITDSIU19025

Le Thi Thu Tra - ITDSIU19058

## **Demo day**

9/1/2021

# Introduction

In this report, we introduce our remake version of the famous Plants vs Zombies© game of Popcap. This is a well-known game, and it is challenging to build it from scratch. We developed this game with the help of JFrame framework of Java. We will give you a brief and concise view about our work. The following sections are organized as follows:

* Section II contains the link to our Github respiratory
* Section III present the game rules and extra features
* Section IV explains the design with UML class diagram
* Section V illustrates the design pattern(s) and principles that we used
* Section VI contain our self-evaluation

# Github repository

Click [here](https://github.com/nmtrang/PvZ) to access our Github repository to have more details about the project through source code.

# Game rule and extra features

## Game rules:

To make this game as simple as possible, we only make the game contain only two rounds: day and night. The initial sun for day round is 2000 and for night round is 3000.

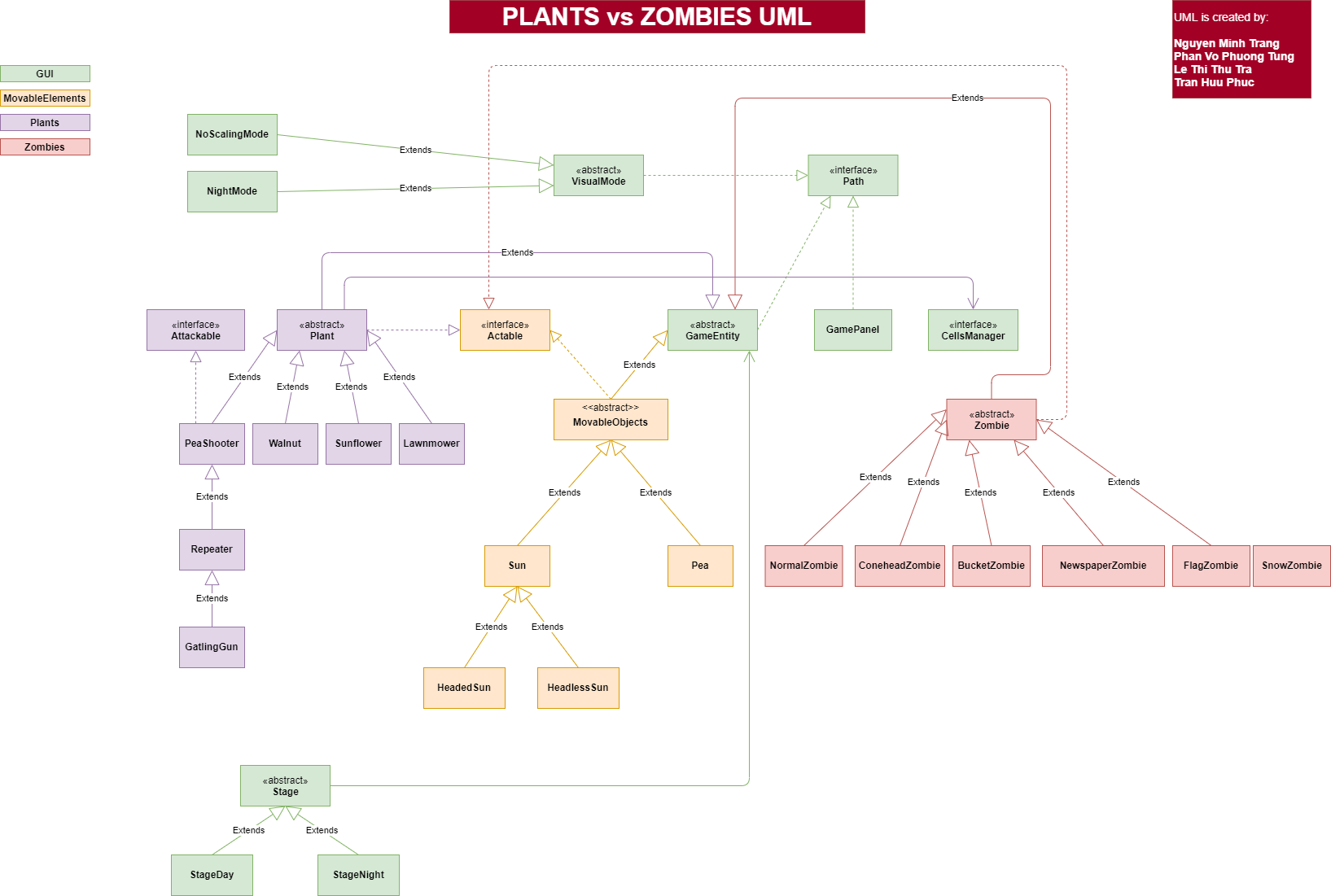
The first round (day round) design with 50 zombies and 6 kinds of zombies (Normal Zombie, Flag Zombie, ConeHead Zombie, BucketHead Zombie and Newspaper Zombie) and 6 kinds of plants (Sunflower, Peashooter, Repeater, Snow Pea, Gatling Pea and Walnut)

The second round (night round) is more complicated. We add another kind of zombie is Snow Zombie (tougher and faster) and two kinds of plants (Sunshroom and Puff Shroom). We also increase the number of zombies we have to kill in order to win to 80 zombies.

## Extra features:

Besides that, we added some extra features. The first one is the Christmas theme. Since it is not long after holiday season so we add Christmas background and some holiday outfits for both zombies and plants. Second, we add another kind of zombie is Snow Zombie. This zombie is tougher, faster and has the appearance of Yeti - a mysterious creature that lives in Himalaya.

# UML Diagram and Design Explanation:



For a better view of our UML, click [here](https://ibb.co/f9L7P0X).

Firstly, we included one basic abstract class to represent all game objects: GameEntity.

This basic class contains common methods like:

* getImage()
* action()

Then, from this base class, we create 3 abstract subclasses to categorizes the entities further:

* Plants
* Zombies
* Movable Objects (Pea, Sun…)

By using polymorphism, we extend each abstract class above to create various concrete classes, each class overwrite the common methods above:

* getImage() : get the appropriate image for each objects
* action(): do the corresponding actions

Constructor

The GUI diagram is the central system in our game that controls the game logic, and this is also where we apply Reflection and Strategy Design Pattern to solve various problems related to switching or adding rounds and visual modes dynamically.

Stage base class declared references to all variables including lists of plants classes, zombie classes, maximum zombie number, visual mode,… Each of those references will be instantiated within the constructors of the concrete subclasses: StageDay and StageNight.

The important thing to notice here is the CellManager and ResourcePath interfaces.

* The ResourcePath interface does two important things, get the right related path (machine independent) and, from that, create and store strings representing relative paths to every resource file.
* CellsManager interface stores the manually measured distance between each cell (as the distances between cells are not uniform) in the map so that we can display plants at suitable positions on the map later.

The main advantage that led us to use interfaces for these things is that all variables declared in an interface are public static final, which matched perfectly to what we need for those variables as they are, literally, just constants.

Each of the visual modes uses the links in ResourcePath interface and gets its needed resources. E.g.: StageNight gets night background image path while NoScalingMode gets the image from day background path.

# Design Pattern/Principles:

The following design patterns and principles are applied in this project, an explanation for their strengths is in Section V.

Implemented patterns/principles:

* DRY principles: we reused code as much as possible in order to shorten the code and make it readable for other coders.
* Defensive design: we tried to avoid errors by utilizing try - catch clauses whenever possible
* Strategy pattern: we were be able to describe encapsulation and abstraction more clearly using this pattern

# Conclusion:

## Overall conclusion:

The project makes good uses of basic Object Oriented programming ideas such as inheritance, interfaces, abstract classes to express the relationship of components of the game. By dividing the project into many levels of abstraction using Strategy design pattern, it is possible to keep the minimum change needed for any upcoming feature/optimization as well as flexibility in the game mechanism, dependencies, and resources. Nonetheless, the lack of a consistent set of principles is also displayed in the code and remains a barrier as the codebase grows larger.

## Strength:

* Know how to categorizes classes and interfaces for the best view
* Amazing teamwork through not only frequently discussing at online meetings but also face-to-face meetings.
* Giving support to other members as much as possible so everyone in the team have the chance to work on different parts of the project
* Members know how to efficiently keep track of the process.

## Subject to improvement:

* Lack of experience and knowledge so it is not that smooth of a game
* Unable to work together remotely (e.g working through Git)
* CellManager cannot apply for various kind of plants (e.g Puffshroom got skewed to the right of frame)
* “Night mode” button did not appeared on screen until moving mouse over it

# 