INTERNATIONAL UNIVERSITY
VIETNAM NATIONAL UNIVERSITY HCMC

January 7, 2021

GROUP PROJECT REPORT

OBJECT ORIENTED PROGRAMMING COURSE

Lecturer: MSc. Le Thanh Son
Lab lecturer: MSc. Nguyen Quang Phu

TOPIC: FLOWERS INVADERS GAME

Group members:

Phan Thi Nhu Yen ITDSIU18048

Nguyen Thi Truong An ITDSIU18002

Contents

1 Introduction

1.1 Background

Since 1970's, people started to take interest in using their computers as an entertainment environment, thus, the multi billion game industry was starting to take shape.

Having the desire not solely to fulfill the requirements for the Object Oriented Programming course we take this semester, but to have a better understanding of the concepts of this prerequisite course, we decided to have a go and create a game of our own.

1.2 Motivations

This Flowers Invaders game is a remake of the original Space Invaders game from 1978 which is a very attractive game for any ages peoples.

The player moves around the galaxy and protect the galaxy from various enemies. They attacks the player with various versions like eggs or any kind of bullet. The player shoots and destroy the enemies and save the galaxy, the score are being increased automatically by destroying each enemies.

2 Analysis and Designs

2.1 Analysis Game Rules Process

In ours project, the player can move left or right at the bottom of the board and will "watering flowers" just like the idea of shooting the enemies thus gets accordingly 100 scores for each normal "watering".

Scores addition upto 6000 points if the player destroyed a boss that has the health of 30 lives.

There is also a bonus plane for the player to get the desire scores of extra 5000 points.

The player has 3 lives and is going to watering 12 stages of flowers in total to complete and win the game, there are 4 boss levels 3, 6, 9 and 12 - the final stage!

The game is developed for entertainment and enthusiasms. It teaches player to be alert every situation he or she faces, not being hit by whatever "the invaders" drop!

Everyone can play this game because controlling the game is very easy, we implemented it to work on some neighboring keys of the keyboard.

2.2 Features from the original Space Invaders and Extra Features

- GUI for various options from player
- Total 12 levels of flowers to be watering implemented

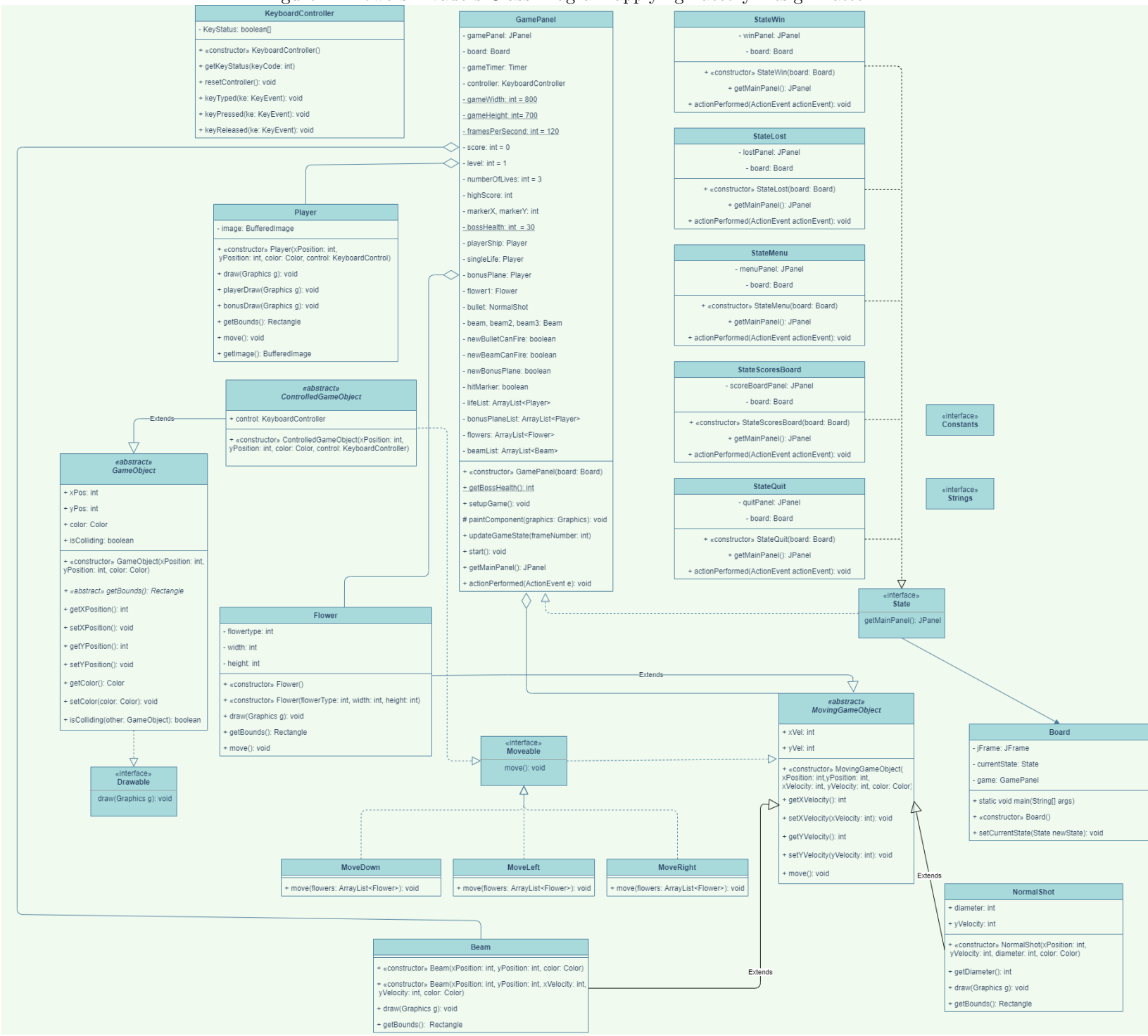
- Attractive game background
- Player as a whale (or various types to be updated)
- Different flower types
- Bonus plane for extra 5000 points
- Player has 3 lives
- High scores automatically saved in file .txt
- Game sounds

2.3 Designs

We indentify the classes that are needed to build the system by analyzing the requirements of the game (based heavily on Space Invaders game). Introducing UML class diagrams to model these classes, this is an important first step in defining the system's structure.

The UML enables us to model, via **class diagrams**, the classes in this "Flower Invades" game and their interrelationship.

Figure 1: Flowers Invaders Class Diagram applying Factory Design Pattern



Link to the class diagram:https://github.com/nhuyen183/OOP_FlowersInvaders/blob/main/Flowers%20Invaders%20UML.pdf

3 Technologies and Programming Languages

As for the programming language, we use JAVA - an object-oriented programming language that is widely used for different purposes and using Factory and Decorator design patterns as well as the SWING framework.

Java Swing is an application program interface or API, associated with Java Programming Language. Swing is a set of rich libraries used to create GUIs or Graphical User Interfaces and is platform-independent because it is completely written in Java.

3.1 Design Classes

In this project, the

3.2 Implementation

```
// Flower.java
import javax.swing.*;
import java.awt.*;

public class Flower extends MovingGameObject
{
       private int flowertype, width, height

       //Constructor
       public Flower()

       @Override
       //Override method from drawable interface to draws flower
       public void draw(Graphics g){ ... }

       //Override method from abstract GameObject class to gets the
           hitbox for normal flowers
       public Rectangle getBounds(){ ... }

       //Override method from Moveable interface used to move all
           flowers
       public void move(){ ...}
}

//Player.java
import Controller.KeyboardController

import javax.swing.*;
```

```
import java.awt.*;
import java.awt.image.BufferedImage;

public class Player extends ControllerGameObject
{
	private BufferedImage image;

	//Constructor
	public Player()

	//Draw
	//Method to draw player objects
	public void playerDraw(Graphics g){...}

	//Method to draw bonus plane
	public void bonusDraw(Graphics g){...}

	//Override method to gets hit box for object
	@Override
	public Rectangle getBounds(){...}

	//Method to move objects
	public void move(){...}

	public BufferedImage getImage(){...}

	@Override
	public void draw(Graphics g){
}}

//GameObject.java
import java.awt.Color;
import java.awt.Rectangle;

public abstract class GameObject implements Drawable
{
	int xPos, int yPos;
	Color color;
	boolean isColliding;

	//Constructor for any game object
	public GameObject(){...}

	public astract Rectangle getBounds();

	//Gets the X position of any object
	public int getXPosition(){ return xPos;}

	//Gets the Y position of any object
	public int getYPosition(){return yPos;}
```

```
        //Gets the color of any object
        public Color getColor(){return color;}

        //Sets the X pos
        public void setXPosition(int xPosition){...}

        //Sets the Y pos
        public void setYPosition(int yPosition){...}

        //Sets the color
        public void setColor(Color color){...}

        //Method to checks if the hitboxes of any two objects
        public boolean isColliding(GameObject other)
{
        isColliding = other.getBounds().intersects(this.getBounds());
        return isColliding;
}}

//ControlledGameObject.java
import Controller.KeyboardController

import java.awt.

public abstract class ControlledGameObject extends GameObject implements
    Moveable
{
        KeyboardController control;

        //Contructor for any controllable object
        public ControlledGameObject(){...}
}

//MovingGameObject
import java.awt.*;

public abstract class MovingGameObject extends GameObject implements
    Moveable
{
        int xVel;
        int yVel;

        //Constructor for any non controllable object
        public MovingGameObject(){...}

        //Gets and sets for every part of the contructor
        public int getXVelocity(){return xVel;}

        public int getYVelocity(){return yVel;}
```

```
        public void setXVelocity(int xVelocity){...};

        public void setYVelocity(int yVelocity){...}

        //Method used to move non controllable objects
        public void move()
        {
                this.xPos += xVel;
                this.yPos += yVel;
        }
}

//Beam.java
import java.awt.*;

public class Beam extends MovingGameObject
{
        //Flowers shot random beams, constructor for Beam
        public Beam(){...}

        //Override method used to draw a beam
        @Override
        public void draw(Graphics g){...}

        //Override method used to get the hit box of a beam
        @Override
        public Rectangle getBounds(){...}
}

//State.java
import javax.swing.*;

public interface State{
        JPanel getMainPanel();
}

//Board.java
import java.swing.*;

//Full code of this main class implemented swing framework that we used
    in the project to build GUI and apply Factory design patterns
public class Board extends JFrame
{
        private GamePanel game;
        private JFrame jFrame;
        private State currentState;

        public void setCurrentState(State newState){
                this.jFrame.getContentPane()..removeAll();
```

```
                this.jFrame.repaint();
                this.currentState = newState;
                this.jFrame.getContentPane().add(this.currentState.getMainPanel());
                this.jFrame.pack();
        }

        public Board()
        {
                //Add text to title bar
                super("Flower Invaders");

                //Make sure the program exists when the close button is
                    clicked
                this.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
                this.jFrame = new JFrame("Flowers Invaders GUI");
                this.jFrame.setPreferredSize(Constants.SCREEN_SIZE);
                this.jFrame.setDefaultCloseOperation(3);
                this.jFrame.getContentPane().setLayout(new
                    BoxLayout(this.jFrame.getContentPane(), 1));
                this.jFrame.setResizeable(false);
                this.jFrame.pack();
                this.jFrame.setVisible(true);
                this.setCurrentState(new StateMenu(this));

                //Create an instance of the Game class and turn onn
                    double buffering to ensure smooth animation
                game = new GamePanel(null);
                game.setDoubleBuffered(true);

                //Add the Breakout instance to this frame's content pane
                    to display it
                this.getContenPane().add(game);
                this.pack();
                this.setResizable(false);
                this.setLocationRelativeTo(null);

                //Start the game
                game.start();
        }

        public static void main(String[] args)
        {
                java.awt.EventQueue.invokeLater(new Runnable(){
                @Override
                public void run(){
                        new Board().setVisible(false);
                }
        });
        }
}
```

```
//GamePanel.java
import Controller.KeyboardControlller;
import sun.audio.AudioPlayer; //package for game sounds
import sun.audio.AudioStream;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner; //for storing highest score in .txt file

public class GamePanel entends JPanel implements State, ActionListener
{
        private JPanel gamePanel;

        private Board board;

        private Timer gameTimer;

        private KeyboardController controller;

        //Controls size of game window and frame
        private final int gameWidth = 880;
        private final int gameHeight = 700;
        private framesPerSecond = 120;

        //Added Counters
        Random r = new Random();
        private int score = 0;
        private int level = 1;
        private int numberOfLives = 3;
        private int highScore;
        private int markerX, markerY;
        private static int bossHealth = 30;
        File f = new File("Highscores.txt");

        //Added Objects
        private Player playerShip;
        private Player singleLife;
        private Player bonusPlane;
        private Flower flower1;
        private NormalShot bullet;
        private Beam beam, beam2, beam3;

        //Added Booleans
```

```
private boolean newBulletCanFire = true;
private boolean newBeamCanFire = true;
private boolean newBonusPlane = true;
private boolean hitMarker = false;

//Added Array Lists
private ArrayList<Player> lifeList = new ArrayList();
private ArrayList<Player> bonusPlaneList = new ArrayList();
private ArrayList<Flower> flowers = new ArrayList();
private ArrayList<Beam> beamList = new ArrayList();
private ImageIcon background = new
    ImageIcon("images/backgroundSkin.jpg");

//Added Audio files and streams
private File bulletSound = new File("sounds/bulletSound.wav");
private File levelUpSound = new File("sounds/levelUpSound.wav");
private File deathSound = new File("sounds/deathSound.wav");
private File hitmarkerSound = new
    File("sounds/hitmarkerSound.wav");
...

//METHODS
//Used in the Flower class to help with the draw method for the
    boss
public static int getBossHealth(){return bossHealth;}

//SETUP GAME
public final void setupGame(){
//Sets flowers for normal levels
if(level != 3 \&\& level != 6 \&\& level != 9 \&\& level != 12){
       //Six rows
       for(int row = 0; row < 6: row++){
              //5 columns
              for(int column = 0; column < 5; column++){
                     //flower speed will increase each level
                     flower1 = new Flower((20 + row * 100)), (20
                         + (column * 60)), level, 0, column,
                         null, 40, 40);
                     flowers.add(flower1);
              }
       }
}
if(level == 3 || level == 6 || level == 9 || level == 12){
       AudioPlayer.player.start(bossSoundAudio); //plays boss
           sound
       flower1 = new Flower(20, 20, 3, 0, 100, null, 150, 150);
           //flowertype of boss = 100
       flowers.add(flower1);
}
//Reset all controller movement
```

```
        controller.resetController();

        //Sets the player's object values
        playeShip = new Player(270, 605, null, controller);

        //Sets the life counter
        for(int column = 0; column < numberOfLives; column++){
                singleLife = new Player(48 + (column * 20), 10,
                    Color.WHITE, null);
                lifeList.add(singleLife);
        }
}

//PAINT
@Override
protected void paintComponent(Graphics graphics){
        super.paintComponent(graphics);

        //Sets background image
        background.paintIcon(null, graphics, 0, -150);

        //makes a string that says "+100" on every flower hit
        if(bullet != null){
                if(hitMarker){
                        graphics.setColor(Color.WHITE);
                        if(level != 3 \&\& level != 6 \&\& level != 9 \&\&
                            level != 12){
                               graphics.drawString("+100", markerX + 20,
                                    markerY -= 1);
                        }else {
                        graphics.drawString("-1", makerX + 75, markerY +=
                            1);
                        }
                }
        }
        //Draws the player's ship
        playerShip.playerDraw(graphics){

        //Draws flowers
        try{
                for (int index = 0; index < flowers.size(); index++){
                        flowers.get(index).draw(graphics);
                }
        }catch(IndexOutOfBoundsException e){}

        //Draw a bullet on space bar press
        if(controller.getKeyStatus(32)){
                if(newBulletCanFire){
                        bullet = new NormalShot(playerShip. getXPosition()
                            + 90, playerShip.getYPosition() -20, 0,
```

```
                    Color.BLUE);
                AudioPlayer.player.start(bulletSoundAudio);
                    //plays bullet sound
                newBulletCanFire = false;
            }
        }
        //Only attempts to draw bullet after key press
        if(bullet != null){
            bullet.draw(graphics);
        }

        //Generates random beams shot from flowers

        //Generates beams at a faster rate for boss

        //Draws the generated beams
        for(int index = 0; index < beamList.size(); index++){
            beamList.get(index).draw(graphics);
        }
        //Generates random bonus plane

        //Draw bonus plane

        //Sets all the displays(score, life counter, level, highestscore
            and a health display for boss level
}

        //UPDATE GAME STATE
        //logic for the collisions and game workflows
        public void updateGameState(){...}

        //GAME PANEL
        public GamePanel(Board board){...}

        //Method to start the Timer that drives the animation for the
            game
        public void start(){...}

        public JPanel getMainPanel(){..}

        public void actionPerformed(ActionEvent e){...}
```

In the project, there is a factory design pattern applied. Related to the state of the game!

We also built basic graphical user interface for the game having 3 buttons as follows

Figure 2: Flowers Invaders Class Diagram applying Factory Design Pattern

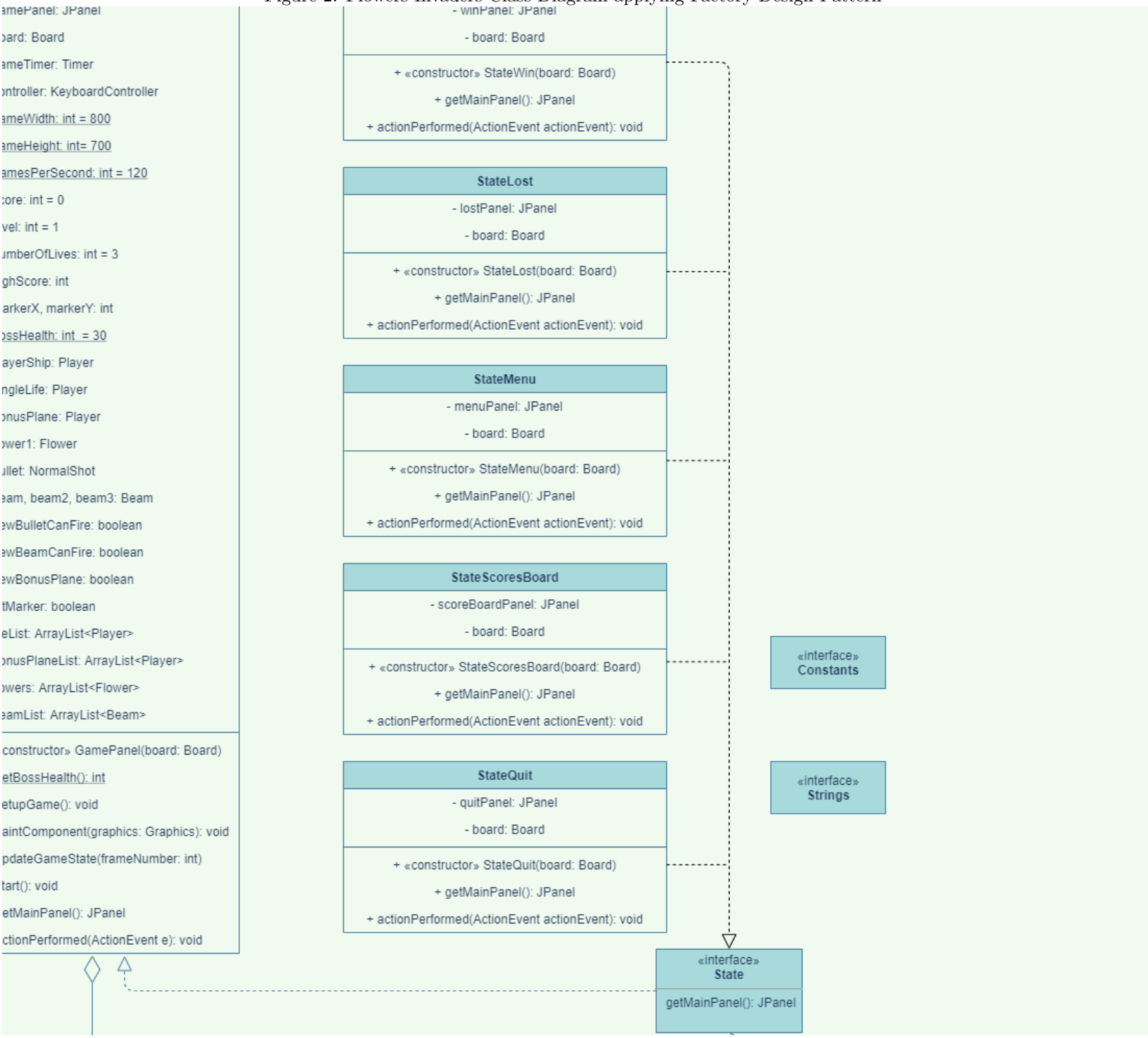


Figure 3: Basic GUI for the game

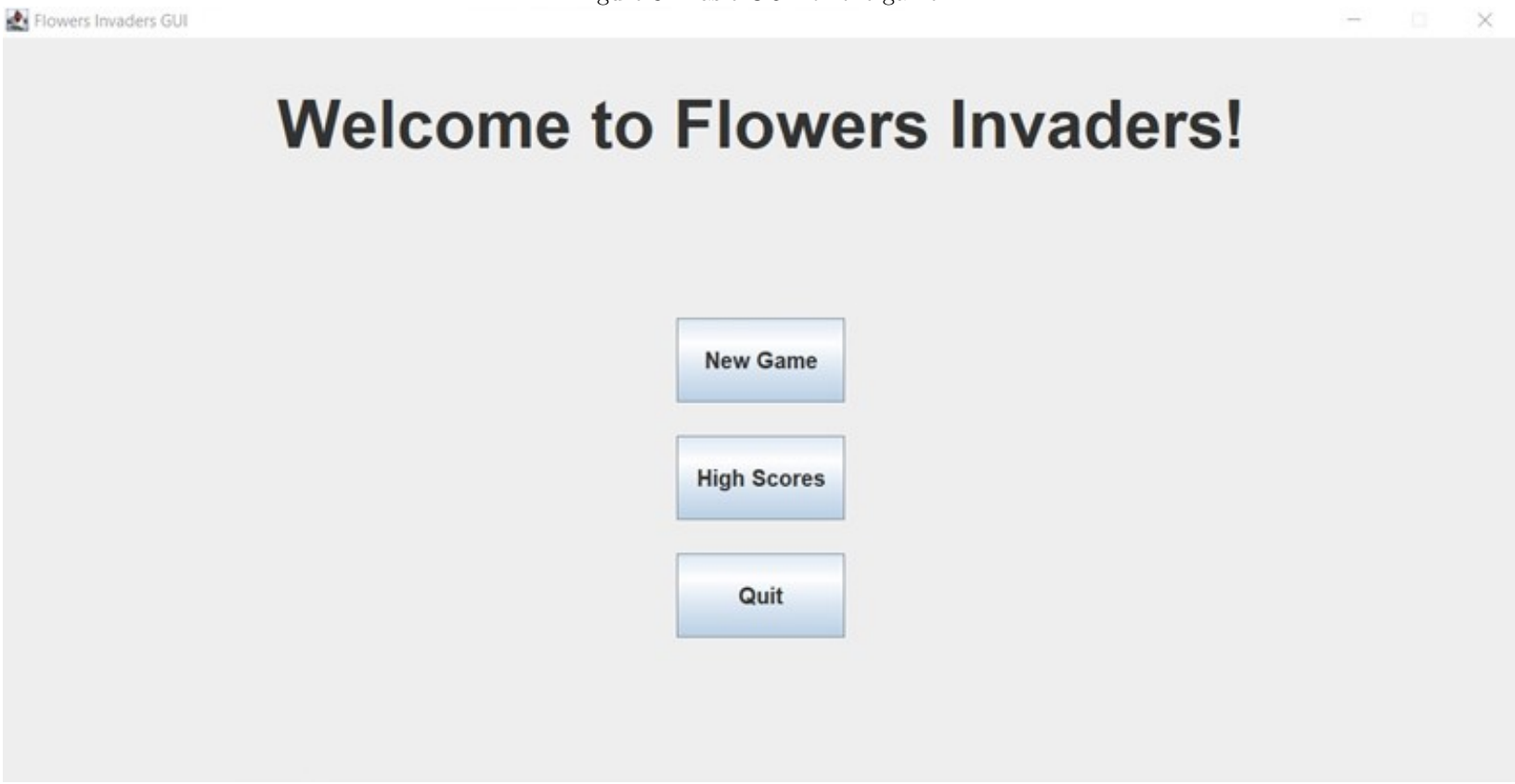
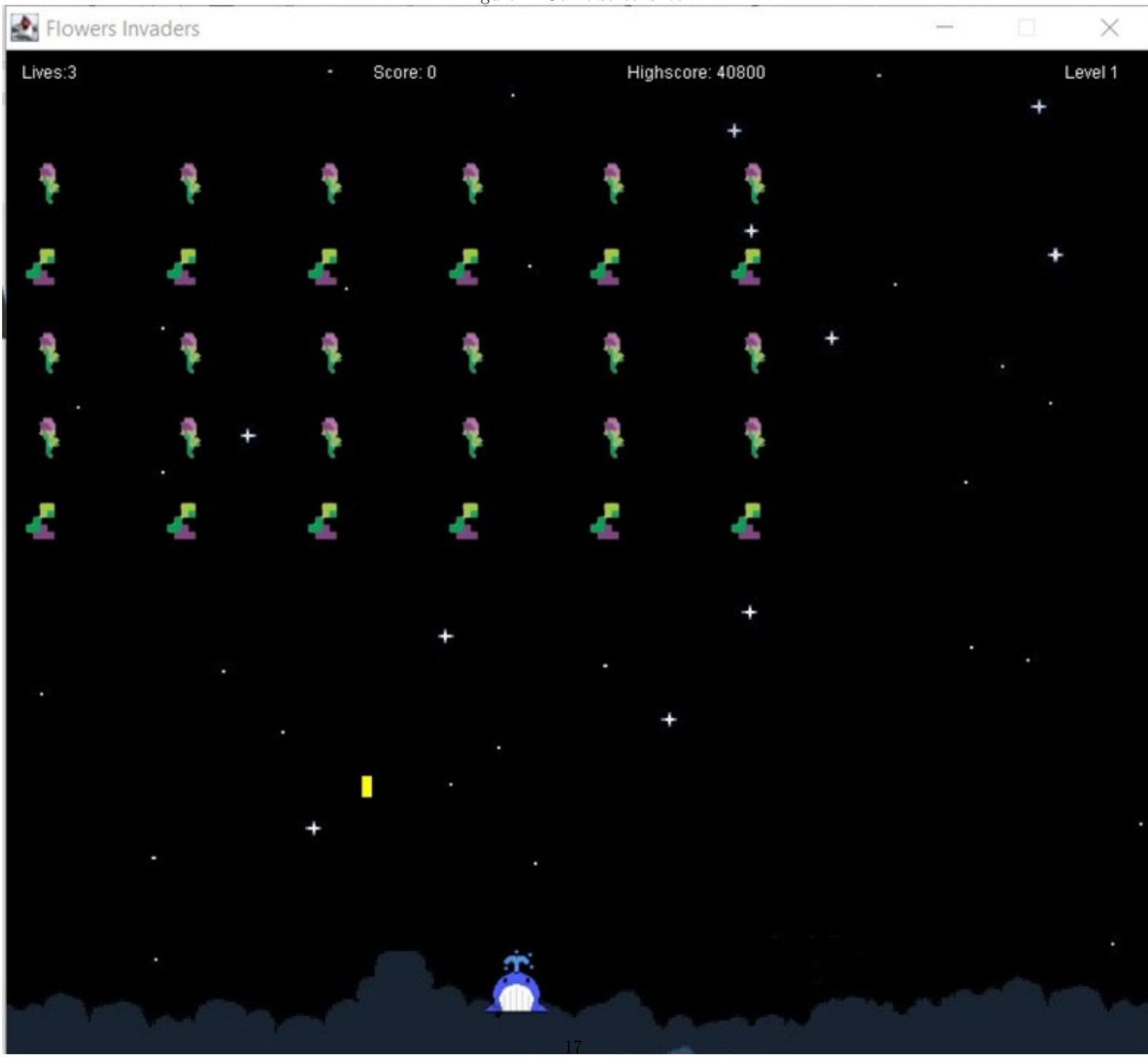


Figure 4: Game screenshot



3.3 Github Project's Link

https://github.com/nhuyen183/OOP_FlowersInvaders

4 Demonstration

IN CLASS.

5 Features to be updated

- Player can choose object they like (whale, spaceship,...)
- Flower can actually bloom by animation...

6 Conclusion

- The project allow us to gain practical experience of the design, prototyping, testing and evaluation stages of application using object-oriented approach.
- We have the chance to pratice programming a lot more and gain practical experience of using object-oriented programming languages.
- We also getting familiar with version of control management: github and gain better understanding of application development in theory and practice.
- Develop communication skills and growing creative thinking and logical thinking in imagination approach.