

INVADERS

CI401 Java Programming Project

Jazer Barclay

STUDENT ID: 20837308

CI401 – INTRODUCTION TO PROGRAMMING

28TH APRIL 2021

Table of Contents

<i>Introduction</i>	<i>1</i>
<i>How to run the code.....</i>	<i>1</i>
<i>The code explained.....</i>	<i>1</i>
An overview of the game	1
Logging.....	2
Score return interface	3
Vectors.....	3
GameObject, Entity, Player and Enemy.....	4
The game model.....	5
<i>Testing</i>	<i>6</i>
Play testing.....	6
JUnit testing	7
<i>Documentation</i>	<i>7</i>
<i>Reflection.....</i>	<i>8</i>

Introduction

This report aims to cover the code I wrote to recreate the 1978 space invaders game using the JavaFX library without the use of the university templates or any existing code. This game includes my own custom sprites for the enemies and player, custom audio effects and some adjusted game mechanics.

How to run the code

The provided zip file contains the full Eclipse project with all required dependencies and assets. First extract the zip folder to your eclipse workspace which is by default your home folder. Next use the import wizard to add the project to Eclipse.

Once imported, the project should be ready to run. If you encounter the 'NoClassDefFoundError' exception then please use your own JavaFX library installed on your system. For further help, please refer to this video I made for setting up JavaFX in Eclipse. It can be found via this link: https://youtu.be/N6cZcw8_XtM

The code explained

An overview of the game

I initialised JavaFX in the 'Main' class by extending JavaFX's 'Applications' class. The 'Main' class holds the different screen states the game goes through such as the start screen, invaders game screen and the game over screen. These screens are their own classes with their transitional handlers exposed through setter methods.

On the start screen, the exit button simply quits the application via the 'System.exit()' method and if at any point the close button is pressed, all running threads are stopped and the application comes to a graceful close. Alternatively, if the user presses the start button, the screen is switched to the interactive invaders game screen.

Java Programming Project - Invaders Game

The game screen contains the model, view and controller (MVC) design for user interaction. The model is in charge of the game's state and entities within it. The view is in charge of translating the game model data into graphics the user can see and takes input from the user. The controller is tied to the view and can modify the model based on what inputs were read from the view.

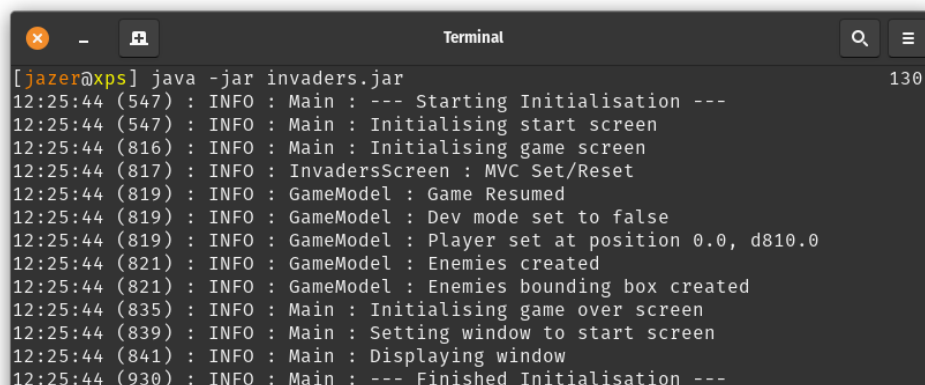
The model also contains the main game loop where each frame is calculated. The loop is run until the game's state is set to stopped and within the model and view skips updating if the state is set to paused. By default when the game model is created, the state is set to running.

If the player is hit with a bullet, the player loses a life. If an invader is hit, they are removed. As invaders are removed, the rate of the game increases resulting in faster moving enemies. Once all invaders are destroyed, they are respawned in at the beginning. These checks are all done in the bullet collision check run every loop.

If the invaders hit the ground or the player loses all their lives, the game completes the main loop and the game over screen is set with the final score via the 'ScoreReturnPromise' interface. The game over screen holds a button to retry which reinitialises the model and switches the scene back to the game screen.

Logging

You will see throughout my code the use of the 'Logger' class. This is a class I created to print information and errors to the terminal output of the application for fault finding and information purposes when testing.



```
Terminal
[jazer@xps] java -jar invaders.jar 130
12:25:44 (547) : INFO : Main : --- Starting Initialisation ---
12:25:44 (547) : INFO : Main : Initialising start screen
12:25:44 (816) : INFO : Main : Initialising game screen
12:25:44 (817) : INFO : InvadersScreen : MVC Set/Reset
12:25:44 (819) : INFO : GameModel : Game Resumed
12:25:44 (819) : INFO : GameModel : Dev mode set to false
12:25:44 (819) : INFO : GameModel : Player set at position 0.0, d810.0
12:25:44 (821) : INFO : GameModel : Enemies created
12:25:44 (821) : INFO : GameModel : Enemies bounding box created
12:25:44 (835) : INFO : Main : Initialising game over screen
12:25:44 (839) : INFO : Main : Setting window to start screen
12:25:44 (841) : INFO : Main : Displaying window
12:25:44 (930) : INFO : Main : --- Finished Initialisation ---
```

The methods in this class are static so they may be accessed without the requirement to create a new object to call them. They are also set as synchronous to prevent race condition errors such as trying to print on both the game loop and the JavaFX thread simultaneously.

The three static methods all call a private method only accessible within the class that takes the parent object, message and exception. The info, warn and error methods simply parse the parameters to the private method making for a single code block to display the text. Should I want to add or change the log string, I can easily modify only the internal method and all three methods will print the new layout.

Score return interface

I needed a way to transition from the interactive 'InvadersScreen' to the 'GameOverScreen' when the loop came to the end. I could have made a button appear when the game loop finished or initialised a new game over screen at the end of the loop but this would be messy and would break the layout of my code. Not only that, I had to get the score to the ending screen back in the 'Main' class. To solve this I used an interface.

This 'ScoreReturnPromise' interface is very simple requiring only a single method 'onReturn(int value)'. This method will perform an action using the value it was parsed.

```
/**
 * An interface to perform an action when the score value is returned
 * @author Jazer Barclay
 */
public interface ScoreReturnPromise {

    /**
     * Perform an action using the value returned
     * @param value
     */
    public void onReturn(int value);
}
```

This solved my problem as the 'GameModel' could take the promise and run the method after the loop; sending the score and functionality to be performed in the 'Main' method to interact with the 'GameOverScreen' class.

Vectors

I needed a simple way to represent object positional and movement data. Rather than repeatedly calculate the data in an 'GameObject' class, I created a simple Vector class that manages a simple X and Y double value.

This class can be initialised with or without the x and y values set in the constructor via overloading. I defined the default case taking an X and Y value when initialised. Then I created a second constructor that takes no arguments however invokes the first with the default values of x being 0 and y being 0.

I then used encapsulation by defining the x and y values as private only allowing access through getters and setters. This helped manage how the values are updated and allows me to check values without breaking implementations in the rest of my code.

To make vector calculations, I created a simple 'add()' method that can take a vector as its argument and returns the sum. This does not modify the current vector as that is the responsibility of the setter methods.

GameObject, Entity, Player and Enemy

For this simple game we have a few objects at play. We have simple objects such as the bullets the player and enemies fire that only require positional and movement data. For the objects that can be killed we require more information such as health, alive status and score values. Finally for the player and enemies, we need to define set movement speeds and their score values.

For this I created a base 'GameObject' class that defines a positional and movement vector, width, height and base movement speed. These are traits all game objects will have when the game is running.

I then extended this in a new class called 'Entity'. An entity is the base class for all objects that can be attacked and destroyed including the enemies and player. The entity class holds health values which set an 'alive' Boolean flag when the health is set to 0. The 'setHealth()' method is responsible for setting the 'alive' value and prevents the health value going lower than 0. Once the health of the entity reaches 0, no amount of increasing the health will reset the alive flag back to true again.

I created the player class which extends 'Entity' in order to set a fixed health, size and base movement speed in its super constructor definition and only requiring the positional values. These values never change throughout the game. The same applies to the different enemies however there are multiple types with their own score values. These types and score values are stored within their respective 'Enemy' class.

For the enemy types, I created an Enum that holds the different enemy types such as the 'bunny', 'loader' and 'squid' types. Should the enemy not be defined, they default to the bunny however if changed in the model via its setter, the type and value can be updated.

```
/**
 * Defines the types of enemies in the game
 * @author Jazer Barclay
 */
public enum EnemyType {
    BUNNY,
    LOADER,
    SQUID
}
```

This is used when the enemies are generated and assigned their row.

The game model

The most complex part of the code is the game model class that holds the state of the game including all game objects and values. I defined the different values as class variables so they may be accessed anywhere within the class. I also generated getters and setters for these so the controller may change values based on keyboard inputs.

For the values that are fixed I used the final variable modifier to prevent changes to their values. Integer values like the maximum rate of the game and player fire cooldown are not values that can be affected during the games runtime.

```
// Static data
private final int MAX_LIVES = 3, MAX_RATE = 205, PLAYER_CD_DURATION = 100, ENEMY_CD_DURATION = 80;
```

I also defined the enemies as a two-dimensional array. This allowed for easier referencing of row and column when drawing the grid of enemies. To loop through them, I nested two 'for' loops first going over the array of arrays and the second going over each value with that inner array. A good example of this is the detection of bullet collisions.

```
544 /**
545  * Checks for bullet collision with the player and enemies
546  * and handles incrementing score based on enemy value, removing the colliding
547  * bullet and increasing the game rate
548  */
549 private void detectBulletCollision() {
550     if (bullets.size() <= 0) return;
551     ArrayList<GameObject> delBullet = new ArrayList<GameObject>();
552
553     // Loop through all enemies
554     for (Enemy[] enemies : this.enemies) {
555         for (Enemy e : enemies) {
556             // If the enemy is already dead, skip
557             if (!e.isAlive()) continue;
558             // For each bullet, check if colliding with the enemy
559             for (GameObject bullet : bullets) {
```

Another issue that I faced was removing the bullets that had collided but was in the middle of a loop of that particular array list. I could check if the bullet was colliding but I couldn't remove it while looping through the bullets. The solution I found was to store a separate list of bullets listed for removal and removing them after the check loop was complete. This removal is done after the bullets are checked for collisions with the enemies and player.

I had some fun with recursion on the checking of the enemies bounds. This controls where the edge of the enemies are and can be seen when running the code! If you press 'P' when the game is in action, you can enable the developer mode which shows the enemies bounding box as well as some other features. when The bounding box hits the edge of the screen, the enemies' movements inverts.

The 'checkBounds()' method does a check if the enemies on its known column are all dead and if so, increments the boundaries inwards. But what if an inner column had already been destroyed? I made a second check that if any are found alive left or right at the end of the check (i.e., a column was found empty, and the bounds were reduced) then it will call itself to check again with the new updated values in case the next column was also destroyed.

Testing

Play testing

As I built the game, the first tests were to draw the objects in the model to the game's canvas. For this I read through the documentation on the JavaFX website and drew a simple box at the position 0, 0. I then modified the 'drawBox()' method to take a game object rather than static values.

When I got player movement working, I had to ensure the player could not travel further than the edge of the screen. Again, I wrote the method to check for the edge collision and encountered some issues such as the player being moved back to the start of the x axis. This happened because I had used the same code for the left detection on the right side of the screen. Modifying to check for each side individually gave me the result I desired.

JUnit testing

When working on the object collision detection for the bullets and entities, I created a JUnit test that performed checks on the methods using a sample of cases. I created a simple truth test where the objects remain apart at first and after moving will collide.

For an edge case, I made one of the objects after moving come into contact with the other with only an overlap of 1. This returned a correct collision when moved too.

Finally, I created a final check where after moving both objects would still not collide and sure enough, they do not collide even after moving.

```
@Test
public void testCollision() {
    // Create 2 new objects that are next to each other but are not colliding
    GameObject obj1 = new GameObject(new Vector(10, 20), new Vector(0, 0), 200, 100, 5);
    GameObject obj2 = new GameObject(new Vector(8, 20), new Vector(5, 0), 200, 100, 5);

    // Ensure the objects are not currently colliding
    if (obj1.isColliding(obj2)) fail("Objects should not yet be colliding");

    // Ensure the objects will collide based on the motion vector set (8 + 5 > 10)
    if (!obj1.willCollide(obj2)) fail("Objects should be colliding");

    obj2 = new GameObject(new Vector(5, 20), new Vector(5, 0), 200, 100, 5);

    // Ensure the objects are not currently colliding
    if (obj1.isColliding(obj2)) fail("Objects should not yet be colliding");

    // Ensure the objects will collide based on the motion vector set (5 + 5 == 10)
    if (!obj1.willCollide(obj2)) fail("Objects should be colliding");

    obj2 = new GameObject(new Vector(0, 20), new Vector(5, 0), 200, 100, 5);

    // Ensure the objects are not currently colliding
    if (obj1.isColliding(obj2)) fail("Objects should not yet be colliding");

    // Ensure the objects will collide based on the motion vector set (0 + 5 < 10)
    if (!obj1.willCollide(obj2)) fail("Objects should be colliding");
}
```

Documentation

I spent a great amount of time after writing the initial code to refactor and reduce the repeated code. In doing so, I started adding Javadoc comments to my now smaller and better explainable methods and statements. These eventually simplified further to simple getter and setter methods with an internal private method that controlled the data. By using the single forward slash and two asterisks above the method, eclipse allowed me to generate descriptions tied to the method they belonged to.

I later discovered that eclipse is able to generate documentation for the project by exporting it as a Javadoc. This created a docs folder within the project's root directory. The result is a collection of html files that describe how the classes and methods within them function based on the comments provided within the code.

As much as this took a long time to go over to ensure accuracy, this has saved me a great deal of time being able to use the code comments as a source for the documentation. And as a result of this, I have taken to ignoring the resulting doc folder in my source control as the information is already tracked via the code.

Reflection

Did I achieve the result I was hoping for? To answer I would have to say no. I originally wanted to recreate the look, feel and most importantly the original game mechanics however I found the scope quite large for the time allocated to complete the game after getting the game to the minimum viable product stage.

Within this time constraint I omitted the UFO enemy that could have provided extra points or maybe an added mechanic for bullet speed. I also removed the limit on one player bullet at a time in favour of a cooldown timer which strays from the original game.

If I was to improve on this implementation of the game, I would create the UFO enemy type with a bonus score value and gives the player a special ability which also increases the enemy difficulty. To add this, I would create a new UFO value in the 'EnemyType' Enum, create a new timer that counts down from a max value to 0. When it reaches 0, the UFO is created and flies across the screen until it leaves the screen. When hit the players fire rate would be increased and 100 points would be awarded.

I would also implement a level system with different enemy speeds, enemy fire rates and player limitations such as speed or fire rate. I could make this read from a file containing the enemies per level or adjust in-game values to make the levels progressively more difficult.

Java Programming Project - Invaders Game

As a final thought, I would implement a score board tied to an online MySQL server that would allow other players to compare their scores. That way it would be fun and interactive for everyone to join in with some friendly competition. Maybe then you could submit your score too.