University of Hertfordshire

School of Computer Science

BSC Computer Science

6COM1056 - Computer Science Project

Final Project Report (April 2019)
On
‘Geometric Defence’

Initials and Surname: I Ishaq
Student Number: 13052271
Supervisor: Julia Goncharenko

ABSTRACT

This report documents the development process that was undertaken during the development of my game 'Geometric Defence'. This is a game that was created using the Eclipse IDE and was written using the Java programming language. The aim of the game is to place towers to prevent the enemies from overrunning your house and reducing you to 0 lives. The game features 5 different towers and runs across 5 maps. Once you have completed one level the next level will automatically load. Each subsequent level gets harder as more and more enemies try to overrun your defences.

The aim of the project was to create a functioning game that can run without any errors and can later be ported to the Google Play Store. As a result of this the report follows the entire development process from initially setting up the aims of the project all the way to delivering the final product.

Moreover, the report will include the research undertaken during development and how this influenced the project that was being designed. Furthermore, the report includes the whole programming phase and the errors and faults that arose during that phase as well the additional improvements and any additional thoughts about the development of the game.

Finally, the report concludes with a discussion on the journey undertaken and what was learnt during this process. Furthermore, this section discusses the issues that were faced while programming in Java and how these experiences have enriched my knowledge about the language.

CONTENTS

FIGURES

1 INTRODUCTION

Tower defence is a subgenre of real time strategy games. There are many different types of tower defence games some have a set path the enemies will follow while others will have you build the path yourself using the towers. In the end the goal is the same stop the enemies from reaching the target.

1.1 BACKGROUND

The purpose of this project is to design and implement a tower defence game. The project allows me to create my own tower defence game, a genre of games that I have been playing for at least a decade. This game was designed to be played on a wide range of devices. Therefore, the game features simple graphics so it can run on older devices without problems. The game will feature 5 different towers and have 3 different levels. After killing a set number of enemies, the player will advance to the next level with each subsequent level increasing in difficulty.

1.2 FEASIBILITY STUDY

Before beginning a big project such as this is it is important to know if the product being built is feasible. To accomplish this a study is conducted before the project plan is built. This is known as a feasibility study and it the first step in creating an efficient work space. During this it is determined if the project can be built and what must be done before you can start.

There are many things that need addressing before development can start. The most important being the language that the project will be coded using. Before this project I had experience in two different languages Python and Java. After much deliberation I decided to use Java as many argue is the best programming language for building desktop applications, android apps and more (Putano, 2017). The second reason Java was chosen since I wanted to later port to the Google Play Store and this makes it easier since many android applications are coded using Java as this is the official language chosen by the android team. (Hook, 2016)

Once the programming language was chosen, I had to find an appropriate IDE (Integrated Development Environment). I researched online and found many different IDEs that I could choose from. The most popular IDE was Eclipse. It is a free open source program that was released in 2001. As a result of it being the number one it was the IDE I chose. (Heller, 2018).

I had to think of an appropriate name for the game and I eventually settled with the name ‘Geometric Defence’. This name was chosen due to the towers I have designed being shapes. After settling on the name and content of the game development could start.

1.3 OBJECTIVES

Before starting a big project, it is important to know your aims and objectives before beginning as this helps guide you in the right direction when you get stuck. It is also important to categorise these objectives into groups. Therefore, the main objectives of the project are to:

- Have 5 different towers which attack the enemies.
- Have at least 3 levels with different themes.
- Have the enemy follow a set path throughout the level.

Additionally, if time permits, I would like to add the following features:

- 2 different upgrade paths for each tower.
- Sound when the towers attack an enemy.

1.3.1 Project Plan

The Gantt chart in Figure 1 shows the original project plan. It shows the original proposed schedule of the project. It shows the actions taking place during a given day and the estimated time to complete the action.

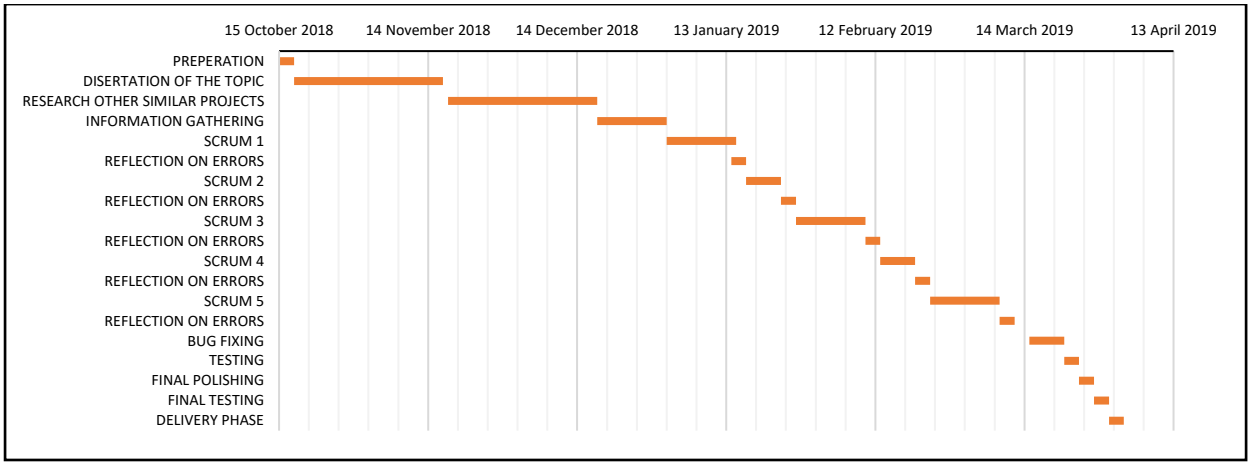

Figure 1 Gantt Chart

The same information can also be viewed as a table which is in Figure 2.

Tasks			
Start Date	**End Date**	**Description**	**Duration (Days)**
15/10/2018	18/10/2018	PREPERATION	3
18/10/2018	18/11/2018	DISERTATION OF THE TOPIC	30
18/11/2018	18/12/2018	RESEARCH OTHER SIMILAR PROJECTS	30
18/12/2018	01/01/2019	INFORMATION GATHERING	14
01/01/2019	14/01/2019	SCRUM 1	14
14/01/2019	17/01/2019	REFLECTION ON ERRORS	3
17/01/2019	24/01/2019	SCRUM 2	7
24/01/2019	27/01/2013	REFLECTION ON ERRORS	3
27/01/2019	10/02/2019	SCRUM 3	14
10/02/2019	13/02/2013	REFLECTION ON ERRORS	3
13/02/2019	20/02/2019	SCRUM 4	7
20/02/2019	23/02/2019	REFLECTION ON ERRORS	3
23/02/2019	09/03/2019	SCRUM 5	14
09/03/2019	12/03/2019	REFLECTION ON ERRORS	3
15/03/2019	22/03/2019	BUG FIXING	7
22/03/2019	25/03/2019	TESTING	3
25/03/2019	28/03/2019	FINAL POLISHING	3
28/03/2019	31/03/2019	FINAL TESTING	3
31/03/2019	03/04/2019	DELIVERY PHASE	3

Figure 2 Project Plan Table

1.4 Report Structure

Chapter two includes the market research undertaken during the project. Each topic is separated into two sub-headings before the conclusion is chapter two point three. The third chapter discusses the requirements and designs of the various elements such as the cells and towers. The fourth chapter shows the implantation of the different aspects of the game such as the enemies and the UI. Chapter five discusses the techniques used while testing and the results of these tests. The project and everything I learnt is concluded in chapter six with any additional work I want to do with the project was discussed in chapter seven.

2 RESEARCH

The following section is the research phase of the project and this began on the 18/11/18.

2.1 MARKET RESEARCH

There are many different tower defence games on many different systems. They range from small online flash games all the way to fully fledge games PC games released on steam. Looking at what these games offer was important to the development of my game since I wanted to include as many features as I could.

2.1.1 Bloons Tower Defense 5

This is arguably the most popular tower game. The game has had multiple releases, and many consider it ubiquitous with the tower defence genre. The aim of this game is to place down many different towers to pop the bloons before they reach the end of the track. Since this is the fifth game in the series it has improved on is predecessors in many ways such as having multiple levels, difficulties, towers, bloons types and upgrade paths. The unique thing about this game is that the longer the game goes on the faster bloons get before giant bloons known as MOAB's start to appear. These take a lot of hits and contain 4 ceramics which contain each of the previous bloons inside it. (Ninjakiwi, n.d.). The version below is the free flash version which can be played on the Ninja Kiwi website. The game however has been released on mobile and steam and those two are considered the definitive versions.

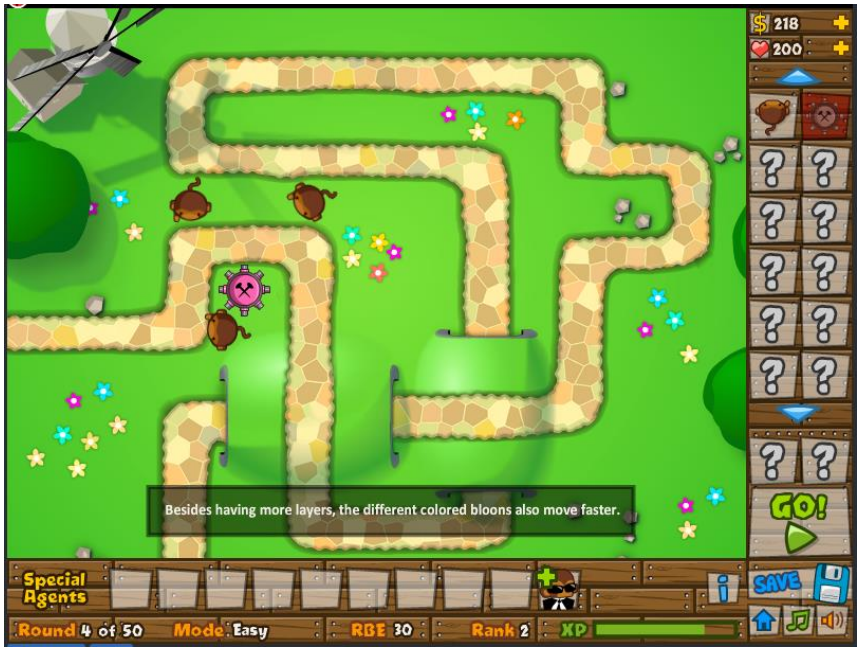

Figure 3 Bloons Tower Defense 5

2.1.2 Desktop Tower Defense

Another popular tower defence game. This was released in 2007 and features hand drawn art. The aim is the same as any other tower defence game kill the enemies before they reach the end of the map. The unique feature about this game is that you can design the path the enemies will take which adds to the replay ability. The game features 6 towers and multiple types of enemies. (Defense, 2007).

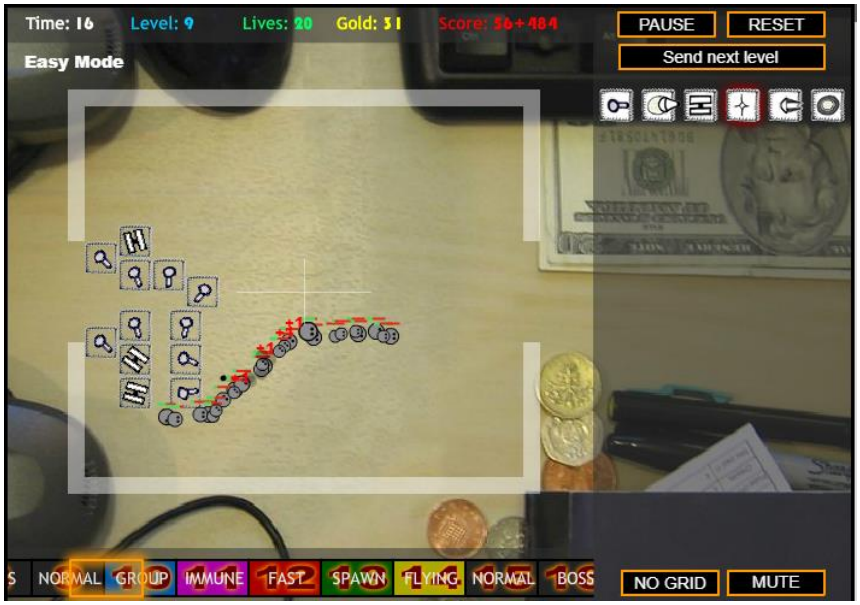

Figure 4 Desktop Tower Defense

2.1.3 Plants Vs Zombies

This is an example of a fully-fledged game released on multiple systems starting with the PC. This was released in 2010 and unlike the previous two this costs £4.25. The game features the same aims as the other two prevent the enemies from reaching your base. The unique thing about this game is that you only have one life per lane meaning you lose the game if more than one zombie enters a single lane. The game features bright and colourful graphics and has 49 different plants which will aid you when trying to defeat the horde trying to eat your brain. (Arts, n.d.)

Figure 5 Plants Vs Zombies

3 REQUIREMENTS & DESIGN

This section focuses on the requirements of the artefact and how the design process was completed. This section began on the 14/01/19 and lasted until 28/01/19.

3.1 REQUIREMENTS

Functional and non-functional requirements are the basic needs for a software to exist. Functional requirements are those that describe what the system should do. On the other hand, non-functional requirements are those that describe how the system works.

3.1.1 Functional Requirements

- Single player game - The should be able to load the game and play without any problems
- Multiple levels - The game should have multiple levels each one increasing in difficulty
- Multiple type of the same tower - The game should handle having multiple types of the same tower on screen at once.
- Five different towers -The game should have five different towers

3.1.2 Non-Functional Requirements

- Learnability - The game should be simple to understand. The user should know what they need to do
- Smooth and responsive - The game should run at a smooth 60 FPS (Frames Per Second)
- Looks - The game should look visually pleasing.
- Efficiency - The game should be efficient with all the information the user would require to play the game on screen.

3.2 DESIGN

To create the art for the game I used Paint 3D. This is a free program which is installed on Windows 10 and it allows users to edit pictures with basic tools such as resize and rotate. I used it create my towers, map tiles and enemies. To accomplish this, I created a blank canvas measuring 26 * 260 and used the brush tool to create the designs I needed. Before creating the designs, I had no idea how to edit/create images apart from at a basic level. This exercise allowed me to get creative with a new program that I had never used and create the assets that will be used in my game.

3.2.1 Layout

The game features two layers one for the ground tiles such as grass, sand etc. The other layer contains that towers and enemies. This allows the background to remain clear when a tower is placed on it or in the case of the road tile enemies walk over it.

3.2.2 Levels

To create the levels, I created a sprite sheet with many different tiles ranging from grass to lava. With these tiles It was possible to create many different environments. The tiles were created using Paint 3D. Once the correct canvas was created the tiles were created using the pen tool. It was important for the game to have multiple different textures as this increases the variety and makes sure that that the player doesn't get bored seeing the same tiles repeatedly.

Figure 6 Tile Sprite Sheet

3.2.3 Towers

As the name of the game suggests the towers this game utilises are geometric shapes. These were created using Paint 3D. Once the canvas was created, I used the built into shape tool to create the outline of the tower and then filled in the colour using the pen tool. To save time and make the code look cleaner I created a sprite sheet for the towers and using eclipse I split that bigger image into separate tiles saving me time from importing multiple images.

Figure 7 Tower Sprite Sheet

3.2.4 Enemies

The original plan of including multiple enemy type could not be realised instead one enemy was created. It was created the same way as the other designs using Paint 3D.

Figure 8 Enemy

3.2.5 UI

The UI elements I needed for this game included an icon for the lives, an icon for the coins you currently have, an icon for the store, an icon for the house the enemy is trying to reach and finally the icon for the trashcan so you can stop selecting a certain tower. All 5 of these were created the same way as the previous designs were created using Paint 3D.

Figure 9 Coin Icon

Figure 10 Store Icon

Figure 11 Lives Icon

Figure 12 House Icon

Figure 13 Trash Can Icon

4 IMPLEMENTATION

In this section I will discuss the implantation process and the steps that were taken on to complete the project. I will discuss the order the project was implemented and why. Furthermore, the technical challenges I faced during implementation will be discussed. This took place later then the original date beginning four weeks after the original date starting on 28/01/19 and lasting until 15/04/19.

4.1 LAYOUT

4.1.1 Frame

The game window was the first thing I worked on as this fed into the other classes such as the enemies and the towers. The first step of this was creating a frame class. This class created the frame that the graphics were going to sit on. To do this I had to extend the JFrame. This created a frame which could be edited. The first thing I did was give it an appropriate name. In this case 'Geometric Defence'. This was done by creating a string variable called title. The next step was creating a variable called size that held the dimensions of the frame. Once this was done, I had to apply these variables the Frame. By setting the title and size a new window with the title 'Geometric Defence' was created. The next step was to make sure the window could not be resized as the art created would not scale properly. This was done using the following code 'setResizable(false);' The window was still not opening in the correct space to solve the following code was used 'setLocationRelativeTo(null);' This ensured the window would open in the centre of the screen no matter the size of the screen. The next step was to close the program whenever the 'x' was pressed. This was done using the follow code. 'setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);'

The next stage was to create a new method call init. This was used to initialise everything. The first thing being setting the layout bounds of the frame. By using the following code 'setLayout(new GridLayout(1, 1, 0, 0));' it made sure the whole frame was drawn on. The next thing was to attach the screen to the frame this was done using the following code 'Screen screen = new Screen(this); add(screen); setVisible(true);' This creates the screen adds it to the frame and makes it visible. The full source code can be seen in Figure 21.

4.1.2 Screen

The Screen class held all the code related to the items be displayed on screen. This class also imports all images and splits the sprite sheets into appropriate size tiles. It is also in charge of loading levels and populating them with the correct tiles. This classes also displays the messages to the player when they complete a level, complete the game or lose too many lives resulting in a game over. Furthermore, it controls the number of enemies required to be killed before the level is complete as well as their speed. Moreover, the class also draws the shop window below the game window. This contains the towers along with their prices so the player can pick a tower a place it. Additionally, it contains the most important method for the game to run, a game loop. It is the overflow control for the entire game and each iteration of the loop is known as a frame. Most modern games run at either 30 or 60 FPS. Since my game runs at 60 FPS the loop completes 60 iterations per seconds. Furthermore, I felt it was important for this information to be visible the player as a result of this both the FPS and the UPS (Updates Per Second) can be seen in the frame next to the name of the game. The full source code can be seen in figures 25 to 29.

4.1.3 MouseTracker

This class contains the code related to the mouse movement. It keeps track when the mouse is being dragged and the new location when the user stops moving it. This is imperative as not tracking the movement of the mouse will make it impossible to play the game. The full source code can be seen in figure 12.

4.2 LEVELS

4.2.1 Cells

This class contains the data for the cells. After creating the variable airID and groundID. A new method called Cell was created this contained six variables this method creates the cell that will be used when drawing the map. This class also contains the code for getting the towers to attack the enemies when they come into range.

When the towers attack an enemy a health bar appears above the target which slowly turns from green to red indicating that the enemy is taking damage. The full source code can be seen in figures 14-16.

4.2.2 Stage

Before the levels could be created the world had to be created. This was accomplished in the Stage class. The first lines of code in this class create the variables needed for this class to function. These are worldWidth, worldHeight and cellSize. These control the size of the world and the size of each cell. Those variables are then used in the define method this creates the world. The physics method is created to allow the enemies to move across the world according to the size of cells. The final method of this class then draws the graphics onto the frame using the information from the next class created Value. The full source code can be seen in figure 31.

4.2.3 Values

This class contains the values all the graphical textures. Since the game is running on two layers. The first layer is the ground layer this contains the values for the ground textures. Each texture is given it own value between 0-7. The second layer is the air layer this contains the values for the texture which will be placed onto the ground textures. Most of this layer will be blank so this was given a value of -1. The goal the enemies are trying to reach is 0. The trash can is 1 and each tower is given a value from 2-6. With these values it was possible to create a level with any of the texture. Using these values, the levels we created and placed in the levels folder. The full source code can be seen in figure 34.

4.2.4 Save

The Save class was created to load the levels into the game. This class first loads the path to the levels folder. Once the folder has been loaded the starts with the first level and gets using that information the level is created. The first part of the level file contains the number of enemies that need to be defeated before the next level can be loaded. The method is surrounded by a catch exception to make sure the program doesn't crash while loading the next level. The full source code can be seen in figure 23.

4.3 TOWERS

The towers were implemented in the cell class. The physics method draws the tower shooting the enemy that come into range. The tower will continue shooting the target until it has lost all its health. Once the enemy has been killed the tower will move on the next target. The full source code can be seen in figures 14-16.

4.4 ENEMIES

The enemy class contains all code regarding enemies. The first part contains the variables such as health and size. This information is used in the subsequent methods. The first being spawnEnemey this as the name suggests spawns the enemies. The next method deleteEnemy removes tells the checkDeath method that enemy has lost all their health. The next important method is physics this gives the enemies the ability to walk. The next part of the method only lets the enemies walk on the road tile. This is done by forcing the player to turn to the next road cell if they run into another tile. The next method in this class loseHealth is used to check if the enemy is dead. It does this by comparing the total health a mob has and the amount they have lost. This is information is passed to the checkDeath method tis gives the player the appropriate amount of coins for the enemy defeated and removes them once they have lost all their health. The final method in this class draws the enemies and their health bars the full source code can be seen in figures 17-20

4.5 UI

This class holds the code regarding the shop. After creating the variables needed the class draws the rectangles which will hold the UI elements. Two small rectangles are created to the left-hand side below the game screen. These two will contain the icons for the lives and the coin icons. Additionally, the rectangles which will hold the towers will be created below the game and filled with the cell icon. The towers are then overlaid onto the first five cells and the last one has the trash can overlaid so the player can delete the tower they have currently selected so they can pick a new one. The full source code can be seen in figures 30-32.

5 TESTING

This section covers the testing process. This began on 16/04/19 and was completed on 23/04/19

Test Case	Description	Data Input (Includes User Interaction)	Expected Result	Actual Result	Pass?
Start program	Make sure the program starts properly without crashing.	N/A	The program opens and the first level starts	The program launched correctly, and the first level started	PASS
Place a tower	Make sure a tower can be placed without crashing	Place a tower	The tower is placed without any issues	The tower was placed without any issues	PASS
The enemy spawn correctly	Make sure the enemies begin to spawn when the level starts	N/A	The level loads and enemies will start to appear a few seconds later	The level loaded and the enemies began to spawn a few seconds later	PASS
The tower shoots the enemy when it comes into range	Make sure the tower will shoot any enemies that come into range	N/A	The tower will start shooting when the enemy come into range	The tower began shooting when the enemy came into range	PASS
The coin value increases when an enemy gets killed	Make sure the coins increase when an enemy is killed	N/A	The coins will increase when an enemy is killed	The coins increased when the enemy got killed	PASS
You lose a life when the enemy reach the house	Make sure lives are lost when the enemy reach the house	N/A	The lives will decrease by one when the enemy reach the house	The lives decreased by one when the enemy reached the house	PASS
You advance to the next stage when the right number of enemies are killed	Make sure the level advances when the right number of enemies are killed	N/A	You advanced to level two when twenty enemies are killed on level one	Level two loaded when twenty enemies were killed	PASS
The game over screen is displayed when all lives are lost	Make sure the game over screen is displayed when all the lives are lost	N/A	The game over screen will appear when the lives reach zero	The game over screen was displayed once zero lives remained.	PASS
The game win screen is displayed when all levels are completed	Make sure the game win screen is displayed when all levels are completed	N/A	The game win screen will appear when you complete all five levels	The game win was displayed when all five levels were completed	PASS

5.1 Testing Results

At the end the game functions pretty much as expected. All the tests were designed to test the functional requirements of the project rather than the non-functional requirements as they are the ones that are required for the game to function correctly and to run smoothly.

6 EVALUATION & CONCLUSION

In this section of the report I will be analysing and discussing the strengths and weaknesses of the project. I will look at the what I did, what I should have done and what could have been done to a higher standard. This section took place after the testing phase starting on the 24-04-19 and finishing on the 29-04-19.

6.1 ARTEFACT EVALUATION

Overall, I am happy with the implantation of the game. the main objectives set out during the planning stage were all met. Originally, I would have been happy with having 3 different levels for the game but implementing new levels proved easy. Since it was so easy to create additional levels, I decided to create 5 levels in total. Each level was harder than the pervious. Any additional work that needs to be done to improve the product have been discussed in section 7.

6.2 TESTING EVALUATION

The testing phase could have done with some additional attention. The tests that were carried out just test the functional requirements as these were the main functions for running the game. If time permitted, I would have liked to also test the non-functional requirements.

6.3 TECHNICAL CONCLUSION

During the development process I have learnt programming is all about planning before you start. If you have planned everything out and you think about what you are doing the process of software development becomes much easier. Choosing Java proved the right choice for a simple game like the one designed here.

6.4 PROJECT CONCLUSION

Overall, I felt that the project was a success, the three main objectives I set out during the planning stage were met. The additional features I wanted to add if time permitted was not possible as I felt twelve weeks was not enough time to complete everything I wanted to accomplish. As the deadline loomed, I felt rushed and found it difficult to stay on track. If we had an additional four weeks, I feel like a better product could have be developed with more polished features as I would have had additional time to prepare and plan every aspect of the game and I could have added the features I missed such as the sound and tower upgrades.

7 FUTURE WORK

The two additional features I wanted to add if time permitted were not able to be implemented as the time constrains prevented me from adding the multiple path upgrades and the sounds.

The game currently has no background music or any sound effects of the towers shooting. Music and sounds are just as important as the graphics when it comes to building a game. Unfortunately, I was unable to implement this during development as I had run out of time.

Furthermore, the game currently has no upgrades for the towers. This is obviously a big omission, allowing the player to upgrade their existing towers to increase stats such as damage and range will increase the level of strategy as you would have the opportunity of having a high level that would be doing considerably higher damage than a few basic towers with no upgrades.

I would also like improve the art design of the game. At the moment most of the tiles and towers are flat shapes adding some depth with shading and other techniques will improve the look and make it more appealing for new users to download and play. Moreover, adding real projectiles that would fire from the tower instead of having a laser line would add to the as these projectiles would take time reaching their target allowing faster enemies to evade slower projectiles. This will add a new layer of strategy as the player would have to find a balance with the faster shooting low damage towers and the slower more powerful towers.

Furthermore, adding variations to the enemies is another feature I would to implement. The game currently only has one enemy and it moves at a set speed and has a set amount of health. Adding slower enemies with more health and faster enemies would add a new layer of strategy requiring the player to think about the tower choice as some would fire slower.

Additionally, implementing a detailed stat screen with the abilities of the towers such as shooting speed, range and damage would be beneficial for the user as they would have a better understanding of what towers are more effective than others.

Furthermore, adding level select screen will make the game more enjoyable as you can pick the level you want to play instead of waiting for it to appear until you beat the previous level to reach it.

Once these improvements have been made, I would not be opposed to port this game to the Google Play Store. Since mobile gaming makes up almost 50% of the global revenue. (Batchelor, 2018).

8 BIBLIOGRAPHY

Ninjakiwi. (n.d.). *Bloons Tower Defense 5*. [online] Available at: https://ninjakiwi.com/Games/Tower-Defense/Play/Bloons-Tower-Defense-5.html [Accessed 20 Nov. 2018].

Defense, D. (2007). *Desktop Tower Defense*. [online] Kongregate. Available at: https://www.kongregate.com/games/preecep/desktop-tower-defense [Accessed 20 Nov. 2018].

Arts, E. (n.d.). *Plants vs Zombies*. [online] Electronic Arts Inc. Available at: https://www.ea.com/en-gb/games/plants-vs-zombies/plants-vs-zombies [Accessed 20 Nov. 2018].

Hook, T. (2016). *I want to develop Android Apps – What languages should I learn?*. [online] Zipcodewilmington.com. Available at: https://www.zipcodewilmington.com/blog/i-want-to-develop-android-apps-what-languages-should-i-learn [Accessed 19 Dec. 2018].

Putano, B. (2017). *Most Popular and Influential Programming Languages of 2018*. [online] Stackify. Available at: https://stackify.com/popular-programming-languages-2018/ [Accessed 20 Dec. 2018].

Heller, M. (2018). *Choosing your Java IDE*. [online] JavaWorld. Available at: https://www.javaworld.com/article/3114167/choosing-your-java-ide.html [Accessed 21 Dec. 2018].

Batchelor, J. (2018). *Global games market value rising to $134.9bn in 2018*. [online] GamesIndustry.biz. Available at: https://www.gamesindustry.biz/articles/2018-12-18-global-games-market-value-rose-to-usd134-9bn-in-2018 [Accessed 29. Apr 2019].

9 APPENDICES

```java
package game;

// Imports
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;

public class Cell extends Rectangle {

    private static final long serialVersionUID = 1L;
    public Rectangle towerSquare;

    //ID number for the ground so each cell can have it's own texture
    public int airID;
    //ID number for the tower so each cell can have a tower on the cell
    public int groundID;
    public int towerSquareSize = 128;


    public int shotEnemey = -1;
    public boolean isShooting = false;

    public Cell(int x, int y, int width, int height, int groundID, int airID) {
        setBounds(x, y, width, height);
        towerSquare = new Rectangle(x - towerSquareSize / 2, y - towerSquareSize / 2, width + towerSquareSize, height + towerSquareSize);
        this.airID = airID;
        this.groundID = groundID;
    }


    // Draws the ground layer
    public void draw(Graphics g) {
        g.drawImage(Screen.tilesetGround[groundID], x, y, width, height, null);
    // Draws the air layer
        if (airID != Value.airAir) {
            g.drawImage(Screen.tilesetAir[airID], x, y, width, height, null);
        }
    }

    public double loseFrame = 1, loseTime = 1 / (double) (Screen.fps);
```

Figure 14 Cell Class Part 1

```java
    // Draws the tower shooting if the enemies come into range
    public void physics() {
        if (shotEnemey != -1 && towerSquare.intersects(Screen.enemies[shotEnemey])) {
            isShooting = true;
        } else
            isShooting = false;

        if (!isShooting) {
            if (airID != Value.airAir && airID != Value.airHouse) {
                for (int i = Screen.enemies.length - 1; i >= 0; i--) {
                    if (Screen.enemies[i].inGame) {
                        if (towerSquare.intersects(Screen.enemies[i])) {
                            isShooting = true;
                            shotEnemey = i;
                        }
                    }
                }
            }
        }
        // While the tower shoots lower the health of the target
        if (isShooting) {
            if (loseFrame >= loseTime) {
                Screen.enemies[shotEnemey].loseHealth(1);
                loseFrame -= loseTime;
            } else {
                loseFrame++;
            }
            // Once the target is dead move onto the next one
            if (Screen.enemies[shotEnemey].isDead()) {
                shotEnemey = -1;
                isShooting = false;
                loseFrame = 1;
            }
        }
    }
```

Figure 15 Cell Class Part 2

```
    // Draws the tower shooting at the enemies
    public void shoot(Graphics g) {
        if (Screen.isDebug) {
            if (airID == Value.airTowerCircle) {
                g.setColor(new Color(0, 0, 0));
                g.drawRect(towerSquare.x, towerSquare.y, towerSquare.width, towerSquare.height);
            }
        }
        if (isShooting) {
            g.setColor(new Color(255, 255, 255));
            g.drawLine(x + width / 2, y + height / 2, Screen.enemies[shotEnemey].x + Screen.enemies[shotEnemey].width / 2, Screen.enemies[shotEnemey].y + Screen.enemies[shotEnemey].height / 2);
        }

    }
}
```

Figure 16 Cell Class Part 3

```
package game;

// Imports
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Rectangle;

public class Enemy extends Rectangle {

    private static final long serialVersionUID = 1L;

    // Int variables
    public int xCoord, yCoord;
    public int health, maxHealth = 200;
    public int healthSpace = 2, healthHeight = 4;
    public int enemeySize = Screen.stage.cellSize;
    public int direction;
    public int enemyID = Value.enemyAir;
    public final int up = 0, down = 1, right = 2, left = 3;

    // Double variables
    public double xd, yd;
    public double enemeyWalk;

    public boolean inGame = false, died = false;

    public Enemy() {

    }
    // Spawns the enemies
    public void spawnEnemy(int enemyID) {
        for (int y = 0; y < Screen.stage.cell.length; y++) {
            if (Screen.stage.cell[y][0].groundID == Value.groundRoad) {
                setBounds(Screen.stage.cell[y][0].x, Screen.stage.cell[y][0].y, enemeySize, enemeySize);
                xd = x;
                yd = this.y;
                xCoord = 0;
                yCoord = y;
                enemeyWalk = 0.0;
                direction = right;
                health = maxHealth;
                break;
            }
        }

        this.enemyID = enemyID;
```

Figure 17 Enemy Class Part 1

```
        inGame = true;
    }

    public void deleteEnemey() {
        inGame = false;
    }

    // Moves the health bar down as the enemy loses health
    public void loseHealth() {
        Screen.life -= 1;
    }

    public double walkFrame = 1, enemyWalkSpeed = 2;

    public void physics(int i) {

        if (direction == up) {
            yd -= enemyWalkSpeed;
            y = (int) yd;
        } else if (direction == down) {
            yd += enemyWalkSpeed;
            y = (int) yd;
        } else if (direction == right) {
            xd += enemyWalkSpeed;
            x = (int) xd;
        } else if (direction == left) {
            xd -= enemyWalkSpeed;
            x = (int) xd;
        }

        enemeyWalk += enemyWalkSpeed;
        if (enemeyWalk >= Screen.stage.cellSize) {
            if (direction == up) {
                yCoord--;
            } else if (direction == down) {
                yCoord++;
            } else if (direction == right) {
                xCoord++;
            } else if (direction == left) {
                xCoord--;
            }
```

Figure 18 Enemy Class Part 2

```
        // Makes the enemies only walk on the road tile.

        try {
            if (direction != down && direction != up) {
                if (Screen.stage.cell[yCoord + 1][xCoord].groundID == Value.groundRoad) {
                    direction = down;
                } else if (Screen.stage.cell[yCoord - 1][xCoord].groundID == Value.groundRoad) {
                    direction = up;
                }
            } else if (direction != right && direction != left) {
                if (Screen.stage.cell[yCoord][xCoord + 1].groundID == Value.groundRoad && direction != left) {
                    direction = right;
                } else if (Screen.stage.cell[yCoord][xCoord - 1].groundID == Value.groundRoad && direction != right) {
                    direction = left;
                }
            }
        } catch (Exception e) {
        }

        if (Screen.stage.cell[yCoord][xCoord].airID == Value.airHouse) {
            deleteEnemey();
            loseHealth();
        }

        enemeyWalk -= Screen.stage.cellSize;
    }
}

public void loseHealth(int amount) {
    health -= amount;
    checkDeath();
}

public void checkDeath() {
    if (health <= 0 && died == false) {
        deleteEnemey();
        died = true;
        getMoney();
        Screen.killed++;
        Screen.hasWon();
    }
}
```

Figure 19 Enemy Class Part 3

```
    // Adds the money to the coin total
    public void getMoney() {
        Screen.coins += Value.deathReward[enemyID];
    }

    public boolean isDead() {
        return died;
    }

    public void draw(Graphics g) {

        // Draws the enemies
        g.drawImage(Screen.tilesetEnemey[enemyID], x, y, width, height, null);

        // Draws their health bars starting green and turning red as they lose health
        g.setColor(new Color(219, 26, 13));
        g.fillRect(x, y - healthSpace - healthHeight, enemeySize, healthHeight);

        g.setColor(new Color(61, 234, 35));
        g.fillRect(x, y - healthSpace - healthHeight, enemeySize * health / maxHealth, healthHeight);

        g.setColor(new Color(0, 0, 0));
        g.drawRect(x, y - healthSpace - healthHeight, enemeySize, healthHeight);

    }
}
```

Figure 20 Enemy Class Part 4

```
package game;

// Imports
import java.awt.Dimension;
import java.awt.GridLayout;
import javax.swing.JFrame;

public class Frame extends JFrame {

    private static final long serialVersionUID = 1L;
    public static String title = "Geometric Defence";
    public static Dimension size = new Dimension(850, 650);

    public Frame() {

        // Sets title of the frame to the string above
        setTitle(title);

        // Sets the size of the window according to the measurements above
        setSize(size);

        // Stops the window from being resized
        setResizable(false);

        // Places the window in the centre of the screen
        setLocationRelativeTo(null);

        // Closes the window when the 'x' in the right hand corner is pressed
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        init();
    }

    public void init() {
        setLayout(new GridLayout(1, 1, 0, 0));

        Screen screen = new Screen(this);
        add(screen);

        setVisible(true);
    }

    public static void main(String[] args) {
        Frame frame = new Frame();
    }
}
```

Figure 21 Frame Class

```java
package game;

// Imports
import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

public class MouseTracker implements MouseMotionListener, MouseListener {

    public void mouseClicked(MouseEvent e) {

    }

    public void mouseEntered(MouseEvent e) {

    }

    public void mouseExited(MouseEvent e) {

    }

    // Selects the tower from the shop once it has been pressed
    public void mousePressed(MouseEvent e) {
        Screen.shop.click(e.getButton());
    }

    public void mouseReleased(MouseEvent e) {

    }

    // Keeps track of the mouse when its being dragged
    public void mouseDragged(MouseEvent e) {
        Screen.mse = new Point(e.getX() - ((Frame.size.width - Screen.myWidth) / 2), e.getY() - (Frame.size.height - Screen.myHeight - (Frame.size.width - Screen.myWidth) / 2));
    }
    // Keeps track of the new location of the mouse when the user stops moving it

    public void mouseMoved(MouseEvent e) {
        Screen.mse = new Point(e.getX() - ((Frame.size.width - Screen.myWidth) / 2), e.getY() - (Frame.size.height - Screen.myHeight - (Frame.size.width - Screen.myWidth) / 2));
    }
}
```

Figure 22 MouseTracker Class

```java
package game;

// Imports
import java.io.File;
import java.util.Scanner;

public class Save {
    public void loadSave(File loadPath) {
        try {
            Scanner loadScanner = new Scanner(loadPath);

            // First loads the number of enemies you need to kill to advance to the next level
            while (loadScanner.hasNext()) {
                Screen.killsToWin = loadScanner.nextInt();
                // Makes sure that the files loading have the correct number of rows and columns which can be found in the stage class
                for (int y = 0; y < Screen.stage.cell.length; y++)
                    for (int x = 0; x < Screen.stage.cell[0].length; x++) {
                        Screen.stage.cell[y][x].groundID = loadScanner.nextInt();
                    }

                for (int y = 0; y < Screen.stage.cell.length; y++)
                    for (int x = 0; x < Screen.stage.cell[0].length; x++) {
                        Screen.stage.cell[y][x].airID = loadScanner.nextInt();
                    }
            }

            loadScanner.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Figure 23 Save Class

```java
package game;

// Imports
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Point;
import java.awt.image.CropImageFilter;
import java.awt.image.FilteredImageSource;
import java.io.File;
import javax.swing.ImageIcon;
import javax.swing.JPanel;

public class Screen extends JPanel implements Runnable {

    private static final long serialVersionUID = 1L;
    public Thread thread = new Thread(this);
    private Frame frame;

    // Int variables
    public static int myWidth, myHeight;
    public static int coins = 10, life = 100;
    public static int killed = 0, killsToWin = 0;
    public static int level = 1, maxLevel = 5;

    // Boolean variables
    public static boolean isFirst = true;
    public static boolean isDebug = false;
    public static boolean won = false;

    public static double fps = 60.0;

    // Makes the original mouse point location at the 0, 0 coordinate
    public static Point mse = new Point(0, 0);

    public static Stage stage;
    public static Save save;
    public static Shop shop;

    // Imports images so they can be used in the game
    public static Image[] tilesetGround = new Image[10];
    public static Image[] tilesetAir = new Image[10];
    public static Image[] tilesetRes = new Image[10];
    public static Image[] tilesetEnemey = new Image[10];
    public static Enemy[] enemies = new Enemy[10];
```

Figure 24 Screen Class Part 1

```
    public Screen(Frame frame) {
        this.frame = frame;
        frame.addMouseListener(new MouseTracker());
        frame.addMouseMotionListener(new MouseTracker());
        thread.start();
    }

    // Keeps track of the number of kills and advances to the next stage when the number has been met. This number can be changed in the level files.
    public static void hasWon() {
        if (killed >= killsToWin) {
            killed = 0;
            won = true;
        }
    }

    public void define() {
        stage = new Stage();
        save = new Save();
        shop = new Shop();

        // Loads the images and splits into separate squares
        for (int i = 0; i < tilesetGround.length; i++) {
            tilesetGround[i] = new ImageIcon("res/tileset_ground.png").getImage();
            tilesetGround[i] = createImage(new FilteredImageSource(tilesetGround[i].getSource(), new CropImageFilter(0, 26 * i, 26, 26)));
        }

        for (int i = 0; i < tilesetAir.length; i++) {
            tilesetAir[i] = new ImageIcon("res/tileset_air.png").getImage();
            tilesetAir[i] = createImage(new FilteredImageSource(tilesetAir[i].getSource(), new CropImageFilter(0, 26 * i, 26, 26)));
        }

        tilesetRes[0] = new ImageIcon("res/cell.png").getImage();
        tilesetRes[1] = new ImageIcon("res/coin.png").getImage();
        tilesetRes[2] = new ImageIcon("res/heart.png").getImage();
        tilesetEnemey[0] = new ImageIcon("res/enemy.png").getImage();

        // Loads the next level from the levels folder
        save.loadSave(new File("levels/level" + level + ".lv"));

        // Creates the number of enemies according to the level
        for (int i = 0; i < enemies.length; i++) {
            enemies[i] = new Enemy();
        }
    }
```

Figure 25 Screen Class Part 2

```
public void paintComponent(Graphics g) {
    if (isFirst) {
        myWidth = getWidth();
        myHeight = getHeight();
        define();

        isFirst = false;
    }

    if (!won) {

        // Sets the colour of the background and draws a black border around the game screen.

        g.setColor(new Color(120, 120, 120));
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(new Color(0, 0, 0));
        g.drawLine(stage.cell[0][0].x - 1, 0, stage.cell[0][0].x - 1, stage.cell[stage.worldHeight - 1][0].y + stage.cellSize);
        g.drawLine(stage.cell[0][stage.worldWidth - 1].x + stage.cellSize, 0, stage.cell[0][stage.worldWidth - 1].x + stage.cellSize, stage.cell[stage.worldHeight - 1][0].y + stage.cellSize);
        g.drawLine(stage.cell[0][0].x - 1, stage.cell[stage.worldHeight - 1][0].y + stage.cellSize, stage.cell[0][stage.worldWidth - 1].x + stage.cellSize, stage.cell[stage.worldHeight - 1][0].y + stage.cellSize)

        stage.draw(g);

        for (int i = enemies.length - 1; i >= 0; i--) {
            if (enemies[i].inGame) {
                enemies[i].draw(g);
            }
        }

        shop.draw(g);


        // Displays the game over screen if the player loses all of their lives

        if (life < 1) {
            g.setColor(new Color(120, 120, 120));
            g.fillRect(0, 0, myWidth, myHeight);
            g.setColor(new Color(255, 255, 255));
            g.setFont(new Font("Courier", Font.BOLD, 16));
            g.drawString("Game Over. The game will close shortly", 10, 20);
        }
```

Figure 26 Screen Class Part 3

```
            // Displays the next level screen if the player successfully completes the level

        } else {
            g.setColor(new Color(120, 120, 120));
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(new Color(0, 0, 0));
            g.setFont(new Font("Courier", Font.BOLD, 15));
            String str = "";
            if (level < maxLevel)
                str += "Level complete. Next level is loading...";
            else
                str += "Congratulations you have won. The game will close shortly.";
            g.drawString(str, 10, 20);
        }
    }
    // Controls how often the enemeies spawn
    public double spawnTime = 90, spawnFrame = 0;

    public void enemeySpawn() {
        if (spawnFrame >= spawnTime) {
            for (int i = 0; i < enemies.length; i++) {
                if (!enemies[i].inGame) {
                    enemies[i] = new Enemy();
                    enemies[i].spawnEnemy(Value.enemeyYellow);
                    break;
                }
            }
            spawnFrame = 1;
        } else {
            spawnFrame++;
        }
    }

    public static double timera = 0;

    public static double winFrame = 1, winTime = 5 * (double) (fps);
```

Figure 27 Screen Class Part 4

```
    // Game loop handles the updating of the screen, fps etc

    public void run() {
        long lastTime = System.nanoTime();
        long timer = System.currentTimeMillis();
        final double ns = 1000000000.0 / fps;
        double delta = 0;
        int updates = 0, frames = 0;

        while (true) {

            long now = System.nanoTime();
            delta += (now - lastTime) / ns;
            lastTime = now;

            // Refresh the screen 60 times a second

            while (delta >= 1) {
                timera++;
                updates++;

                if (!isFirst && life > 0 && !won) {
                    stage.physics();
                    enemeySpawn();
                    for (int i = 0; i < enemies.length; i++) {
                        if (enemies[i].inGame) enemies[i].physics(i);
                    }
                } else if (won) {
                    if (winFrame >= winTime) {
                        if (level >= maxLevel) {
                            System.exit(0);
                        } else {
                            won = false;
                            level++;
                            coins = 10;
                            define();
                        }
                        winFrame = 1;
                    } else {
                        winFrame++;
                    }
                }

                delta--;
            }

            repaint();
            frames++;
```

Figure 28 Screen Class Part 5

```
            // Displays the games FPS (frames per second) and the UPS (Updates per second) in the frame

            if (System.currentTimeMillis() - timer >= 1000) {
                timer += 1000;
                frame.setTitle(Frame.title + " | ups: " + updates + " | fps: " + frames);
                updates = 0;
                frames = 0;
            }
        }
    }
}
```

Figure 29 Screen Class Part 6

```java
package game;

// Imports
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Rectangle;

public class Shop {

    // Int variables
    public static int shopWidth = 7;
    public static int buttonSize = 64;
    public static int cellSpace = 2;
    public static int largeCellSpace = 21;
    public static int iconSize = 20;
    public static int iconSpace = 6;
    public static int iconTextY = 15;
    public static int itemIn = 4;
    public static int heldId = -1, realId = -1;
    public static int[] shopID = { Value.airTowerCircle, Value.airTowerTriangle, Value.airTowerDiamond, Value.airTowerStar, Value.airTowerPentagon, Value.airAir, Value.airTrashCan };
    public static int[] price = { 5, 10, 15, 20, 25, 0, 0, 0 };

    // Creates the rectangles below the game
    public Rectangle[] cell = new Rectangle[shopWidth];
    public Rectangle cellHealth, cellCoins;

    public boolean holdsItem = false;

    public Shop() {
        define();
    }

    // Deletes the item you are holding when you press the trash can
    public void click(int mouseButton) {
        if (mouseButton == 1) {
            for (int i = 0; i < cell.length; i++) {
                if (cell[i].contains(Screen.mse)) {
                    if (shopID[i] != Value.airAir) {
                        if (shopID[i] == Value.airTrashCan) {
                            holdsItem = false;
                            heldId = Value.airAir;
                        } else {
                            heldId = shopID[i];
                            realId = i;
                            holdsItem = true;
                        }
                    }
                }
            }
```

Figure 30 Shop Class Part 1

```java
            // Allows you to place a tower if you have the right amount of money
            if (holdsItem) {
                if (Screen.coins >= price[realId]) {
                    for (int y = 0; y < Screen.stage.cell.length; y++) {
                        for (int x = 0; x < Screen.stage.cell[0].length; x++) {
                            if (Screen.stage.cell[y][x].contains(Screen.mse)) {
                                if (Screen.stage.cell[y][x].groundID != Value.groundRoad && Screen.stage.cell[y][x].airID == Value.airAir) {
                                    Screen.stage.cell[y][x].airID = heldId;
                                    Screen.coins -= price[realId];
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    // Draws the rectangles for the differnet UI elements
    public void define() {
        for (int i = 0; i < cell.length; i++)
            cell[i] = new Rectangle(Screen.myWidth / 2 - shopWidth * (buttonSize + cellSpace) / 2 + (buttonSize + cellSpace) * i, Screen.stage.cell[Screen.stage.worldHeight - 1][0].y
                    + Screen.stage.cellSize + largeCellSpace, buttonSize, buttonSize);
        cellHealth = new Rectangle(Screen.stage.cell[0][0].x - 1, cell[0].y, iconSize, iconSize);
        cellCoins = new Rectangle(Screen.stage.cell[0][0].x - 1, cell[0].y + cell[0].height - iconSize, iconSize, iconSize);
    }

    public void draw(Graphics g) {


        for (int i = 0; i < cell.length; i++) {
            if (cell[i].contains(Screen.mse)) {
                g.setColor(new Color(255, 255, 255, 150));
                g.fillRect(cell[i].x, cell[i].y, cell[i].width, cell[i].height);
            }
            // Fills in each rectangle with its own tower.
            g.drawImage(Screen.tilesetRes[0], cell[i].x, cell[i].y, cell[i].width, cell[i].height, null);
            if (shopID[i] != Value.airAir) g.drawImage(Screen.tilesetAir[shopID[i]], cell[i].x + itemIn, cell[i].y + itemIn, cell[i].width - itemIn * 2, cell[i].height - itemIn * 2, null);
            if (price[i] > 0) {
                g.setColor(new Color(255, 255, 255));
                g.setFont(new Font("Courier", Font.BOLD, 15));
                g.drawString("$" + price[i], cell[i].x + itemIn, cell[i].y + itemIn + 10);
            }
        }
```

Figure 31 Shop Class Part 2

```java
        // Draws the health and coin icons
        g.drawImage(Screen.tilesetRes[2], cellHealth.x, cellHealth.y, cellHealth.width, cellHealth.height, null);
        g.drawImage(Screen.tilesetRes[1], cellCoins.x, cellCoins.y, cellCoins.width, cellCoins.height, null);
        g.setFont(new Font("Courier", Font.BOLD, 15));
        g.setColor(new Color(255, 255, 255));
        g.drawString("" + Screen.life, cellHealth.x + cellHealth.width + iconSpace, cellHealth.y + iconTextY);
        g.drawString("$" + Screen.coins, cellCoins.x + cellCoins.width + iconSpace, cellCoins.y + iconTextY);

        // Draws the item you are currently holding next to your cursor
        if (holdsItem) g.drawImage(Screen.tilesetAir[heldId], Screen.mse.x - (cell[0].width - itemIn * 2) / 2 + itemIn, Screen.mse.y - (cell[0].height - itemIn * 2) / 2
                + itemIn, cell[0].width - itemIn * 2, cell[0].height - itemIn * 2, null);
    }
}
```

Figure 32 Shop Class Part 3

```java
package game;

// Imports
import java.awt.Graphics;

public class Stage {

    // Int variables
    public int worldWidth = 12;
    public int worldHeight = 8;
    public int cellSize = 64;

    public Cell[][] cell;

    public Stage() {
        define();
    }

    // Creates the game window according to the values above (IF CHANGED HERE MAKE SURE YOU CHANGE THE LEVEL FILES TO REFELCT THIS CHANGE)
    public void define() {
        cell = new Cell[worldHeight][worldWidth];

        for (int y = 0; y < cell.length; y++)
            for (int x = 0; x < cell[0].length; x++)
                cell[y][x] = new Cell((Screen.myWidth / 2) - ((worldWidth * cellSize) / 2) + (x * cellSize), y * cellSize, cellSize, cellSize, Value.groundGrass, Value.airAir);

    }

    // Allows the enemies to move across cells
    public void physics() {
        for (int y = 0; y < cell.length; y++) {
            for (int x = 0; x < cell[0].length; x++) {
                cell[y][x].physics();
            }
        }
    }
    // Draws the graphics onto the frame
    public void draw(Graphics g) {
        for (int y = 0; y < cell.length; y++)
            for (int x = 0; x < cell[0].length; x++)
                cell[y][x].draw(g);
        for (int y = 0; y < cell.length; y++)
            for (int x = 0; x < cell[0].length; x++)
                cell[y][x].shoot(g);
    }
}
```

Figure 33 Stage Class

```java
package game;

public class Value {

    // ID Values of the ground textures
    public static int groundGrass = 0;
    public static int groundRoad = 1;
    public static int groundSand = 2;
    public static int groundWater = 3;
    public static int groundLava = 4;
    public static int groundDirt = 5;
    public static int groundSnow = 6;
    public static int groundStone = 7;

    // ID values of the air textures including the towers
    public static int airAir = -1;
    public static int airHouse = 0;
    public static int airTrashCan = 1;
    public static int airTowerCircle = 2;
    public static int airTowerTriangle = 3;
    public static int airTowerDiamond = 4;
    public static int airTowerStar = 5;
    public static int airTowerPentagon = 6;


    public static int enemyAir = -1;
    public static int enemeyYellow = 0;

    public static int[] deathReward = { 2 };
}
```

Figure 34 Value Class