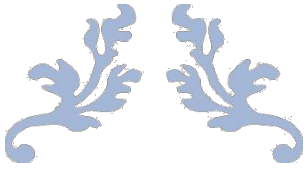
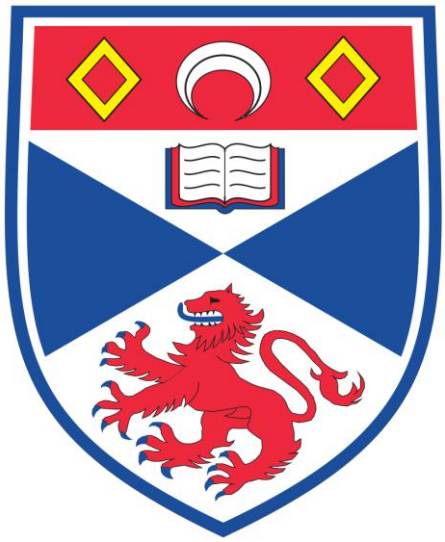


Practical 2

Object Oriented Implementation: Tower Game

CS5001 – Object Oriented Modelling Design & Programming



170009629

pmh20@st-andrews.ac.uk

OCTOBER 17th, 2017

UNIVERSITY OF ST ANDREWS

i. List of Abbreviations/Short-Hands

Unified Modelling Language	UML
Health Points	HP
Speed	Spd
LT	Load Time
Damage	Dmg
Range	Rng
Unlimited	Unlmtd
User Interface	UI

ii. Table of Figures

iii. Table of Contents

1 Introduction

1.1 Aim

The aim of this practical was to demonstrate the knowledge und understanding gained in the area of object-oriented programming. This was done by implementing a number of classes and objects through the creation of a game. The game chosen for this practical was a 'Tower Defence Strategy Game'.

In order to be successful, Super-classes and Sub-classes were to be utilised, with the game able to run whilst following the provided rules. This involved the important element of ensuring that the objects all interact correctly with each other, within a sensible construct of game logic.

1.2 Concept of Tower Defence Games

The concept behind a tower defence game is simply the protection of some territory. It is protected by placing what are known as towers to defend from approaching enemies. This simple premise can be implanted in 1, 2, or 3 dimensions in order to view the outcome.

2 Playing the Game: Instructions

2.1 Game Premise (Story)

Setting: Middle Earth (the world from ' The Lord of the Rings' series).

Disclaimer: The following story is in no way part Tolkien's original writings.

Storyline: During the quest to Mordor to destroy Sauron's one true ring, the members of the 'Fellowship of the Ring' lost Frodo and Sam, who departed for Mordor alone at Argonath. The Fellowship went on to Rohan where they separated further. While Gandalf and Aragorn continued to Edoras, and Merry and Pippin traversed Fangorn Forrest with the Ents, Gimli the Dwarf and Legolas the Elf stayed behind. It is here that they received word that the beautiful Elf City of Rivendell (**figure 1**) was being marched upon by Sauron's armies.

The history between Elves and Dwarves is rich with confrontation, with Dwarves feeling very betrayed and untrusting of the Elves. However, the newfound friendship between Gimli and Legolas was enough to overcome these differences, and they raced back to protect Rivendell, gathering troops along the way, from each of their races alike.

In this game, you the user will play as the combined strategic mind of Gimli and Legolas, who will lead their troops in the City defence against the oncoming enemies. The task is simply to hold the enemies off until reinforcements arrive (perhaps the nearby stubborn Forrest Elves who, due to their differences, do not readily defend 'City Elves').

Figure 1 - The City of Rivendell [1]

2.2 Characters and Weapons

2.2.1 Attacking Enemies (Sauron's Army)

The attacking army of Sauron's consist of 4 main groups, which are explained below.

Rats

Bio: Commonly referred to as "Rats", Wargs are rodent type wolves that are often used as expendable pawns. They are fast, but lack defence (HP).

Stats:
- HP = 1
- Spd = 2

Figure 2 - Warg (Rat) [2]

Elephants

Bio: Mûmakil (or Oliphaunts) are a type of elephant, but of gigantic size. They range from 40 – 50ft in height, and often support large towers on their backs. They are slow but have great resistance to damage attacks.

Stats:
- HP = 10
- Spd = 0.5

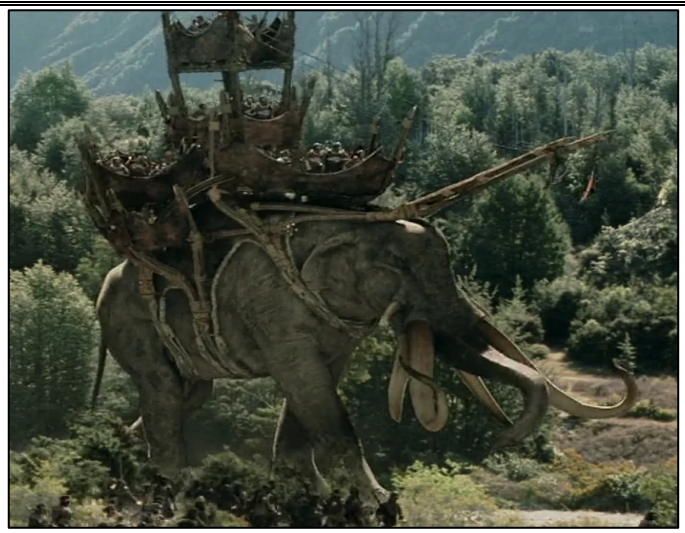


Figure 3 - Mûmakil (Elephant) [3]

Orcs

Bio: Descending from badly tortured elves, and surviving through cruel conditions, Orcs act as the main foot soldiers of Sauron's army. They fight well with great speed and decent health.

Stats:
- HP = 5
- Spd = 3

Figure 4 - Orc foot soldier [4]

Uruk Hai

Bio: Rumoured to be the product of cross-breeding Orcs with men, Uruk Hai are the 'Great Soildier-Orcs', first created in the 3rd age by Saruman. They are both extremely fast and resilient.

Stats:
- HP = 8
- Spd = 5

Figure 5 - Uruk Hai with its creator, Saruman [5]

2.2.2 Ally Reinforcements

Your defending reinforcements consist of 2 main race types and 2 main weapons. These are shown below.

Slingshot

Description: The Slingshots available for use are quick-fire, short-ranged weapons. As the sling requires to be wound, damage is sacrificed for load time by winding less.

Stats:
- LT = 1
- Dmg = 1
- Rng = 40

Figure 6 - Slingshot [6]

Catapult

Description: The Catapult is a large contraption, and can fire at a much longer range and with greater dmg than the Slingshot. However, this power comes at the cost of a far slower load time.

Stats:
- LT = 3
- Dmg = 5
- Rng = 80

Figure 7 - Catapult (loaded here by a troll) [7]

Dwarf Axeman

Description: The dwarves of Middle Earth are proud and stubborn, but are incredibly tough in battle (especially with an axe). However, they must be in close-combat, and cannot repeatedly kill without break.

Stats:
- LT = 4
- Dmg = 7
- Rng = 3

Figure 8 - Axe-wielding Dwarf [8]

Elf Bowman

Description: The Elves are excellent with ranged weapons, particularly the long bow. With the impeccable 'Elf-Vision', they can shoot longer than any battle-field, however, damage is reduced at this long range.

Stats:
- LT = 2
- Dmg = 3
- Rng = Unlmtd

Figure 9 - Elf Archer in both stances (rest and ready) [9]

2.3 Game Rules

- To win, player must complete all 20 waves of enemies (consisting of 25 time-steps each), by defending the territory: 'Rivendell'.
- Player loses when a single enemy crosses the territory border.
- Enemies spawn at the start of each wave, increasing in difficulty with each wave.
- Player starts with buying an initial Tower with starting balance.
- Player earns money by killing enemies. See **figure 10** for variation of earnings.
- Player can purchase a **MAX** of 1 tower per wave, so choose wisely (save when needed). See **figure 10** for variation of prices.
- Player may choose where to place the tower (this is where range is important).
- Player inputs are not case-sensitive, however, if an invalid letter is pressed, you must wait until the next wave to try again (a severe penalty for the mistake).

Table of Finances	
Enemy	**Earnings from Killing**
Rat	£5
Elephant	£40
Orc	£50
Uruk Hai	£100
Tower	**Price**
Slingshot	£300
Catapult	£500
Dwarf Axeman	£1200
Elf Bowman	£1500

Figure 10 - Table of financial earnings and costs

3 List of Parts Completed

3.1 Basic Requirements of the Practical

All basic requirements were completed, and are listed as follows:

- Implement all classes and methods from UML.
- Each Tower has a monetary cost, and can be purchased using bank balance.
- The method **toString()** was overridden to aid in debugging with useful information.
- The game runs correctly as intended, and passes all JUnit tests and style-check.

3.2 Practical Enhancements

All 'possible enhancements' suggested were completed, and are listed as follows:

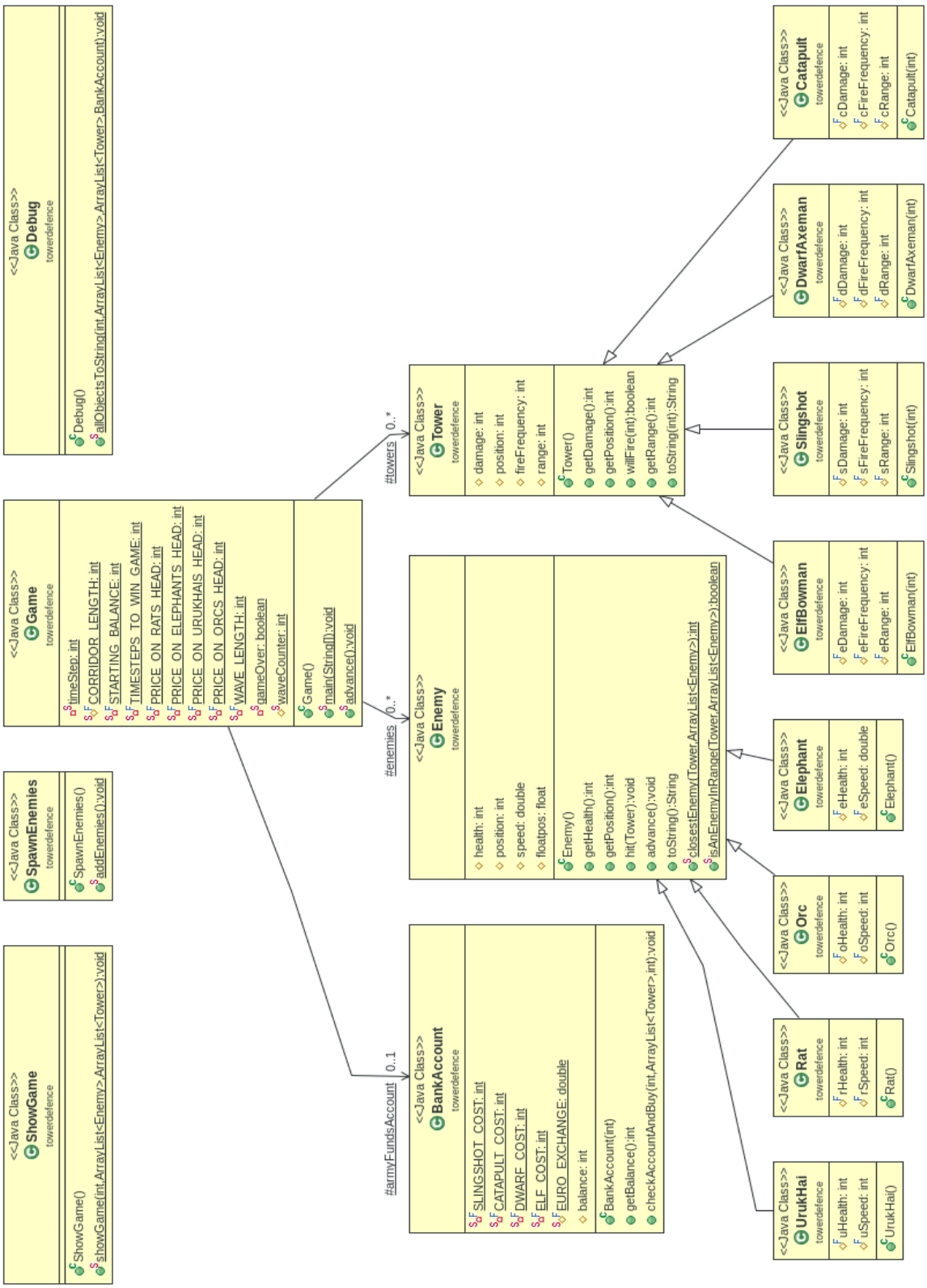
- Player earns money for each enemy kill (different for each enemy). The earned money is used to buy more Towers. Refer to **figure 10** for all financial queries.
- Supplementary to the original Enemies/Towers, 2 additional Enemies types were added, and 2 additional Towers types were added, each with unique characteristics.
- A terminal based User Interface (UI) was designed, in order to depict the game unfolding.
- Enemies were introduced automatically.

3.3 Additional (Further) Enhancements

Some further enhancements were completed in order to make the game operate better. These were as follows:

- A 'range' for each Tower was introduced, meaning they not only could not shoot behind them (to the right of their position), but also could not shoot an enemy over a certain distance away (set by each Tower's range characteristic).
- Rather than simply letting the player make decisions at the start, then watch the UI play out the game, a number of waves were introduced. In between each wave, the player is able to access the shop and purchase further allies/weapons. They can also view the previous wave, in order to strategically prepare for the next one.

4 Updated Unified Modelling Language (UML)



5 Design Decisions and Justification

5.1 Enemies and Enemy Spawning

2 additional enemies were chosen to give the game a more realistic feel, like a real battle. Wide ranges of character attributes were used to ensure the player had to think strategically at times, which also increases the difficulty of the game.

Enemies were spawned in waves, to give an effect of 'game levels'. This allowed a simple way of increasing the difficulty each time so that wave 20 (the last round) was the hardest.

Although new enemies would spawn only at the star of each wave, the quantity of each type was randomly selected (between suitable values for each enemy that allowed a natural feel). This allowed every game to be unique when replaying. The amount selected would also be subject to scaling, dependant on the current wave.

In the event that the player cannot clear all enemies before the wave is over, the enemies left on the battlefield are **NOT** removed, and instead carry over into the next wave. This is more realistic, and gives the game a new strategic element. Rather than simply playing to get through the wave (and waiting for an enemy reset), the player must plan for future waves, which tests strategic endurance.

5.2 Tower 'Shop'

The Tower shop (**figure 11**) is open for business only before each wave begins, when there is time between the enemy advances. This was so the game didn't have to be stopped too frequently just to buy towers, and added the challenge of needing to make it to the end of the wave.

Only 1 Tower can be bought per wave, which means careful thought must go into which one (or to even buy nothing and save up instead). This was done for safety, because depending on the partially random enemy generation, a player may earn too much money at once, and this prevents the buying of many Towers (which would make the game overly easy).

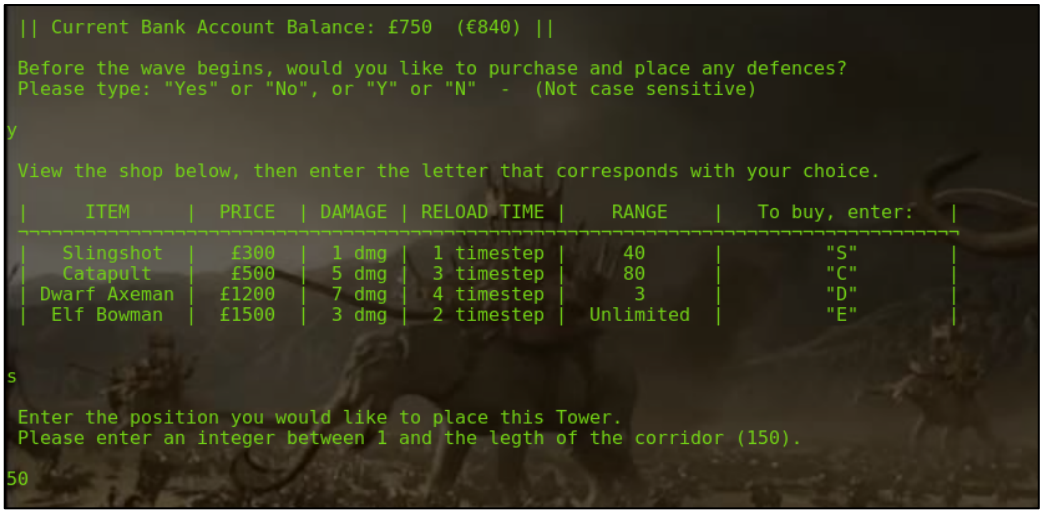


Figure 11 - Image of the in-game Tower Shop, printed in the terminal

5.3 Range

This was a very important feature for making the game function like a real game. Without a range attribute for the towers, there would be a serious flaw in the game's playability. With strategy in mind, with no range restrictions, the logical choice would **always** be to place the towers at the very end of the corridor in front of the territory, as they will have the longest time to kill enemies with no worries of the 'no backwards shooting' rule, and no downsides. This takes away the importance of the player's choice in positioning. Therefore a range was introduced, resulting in a Tower only being able to hit an enemy within its specified maximum range.

5.4 Enemy Targeting

The game was chosen to be more 2-dimensional than 1-dimensional, representing a battlefield more than a corridor. This means that Enemies (or Targets) in the same position (x-axis), are not simply on top of each other, but are instead shoulder-to-shoulder in a longitudinal line (representing an army squad).

For this reason, it makes sense that a Tower can only damage one Enemy at a time. To decide which one, an algorithm was made to calculate the closest enemy on the Euclidean plane. The Tower then targets and fires on this 'closest Enemy'.

5.5 User Interface Display

For display the game, a terminal UI was designed. This used text output to give instructions/ask questions, take user input decisions, and then display the resulting wave timestep-by-timestep for the player to inspect.

It was decided that for ultimate ease of viewing the stats of every Tower and Enemy currently in the map would be printed, followed by a visual display of the positions. For clarity, the visual display also indicated the Tower/Enemy type by showing its first letter, and also its current damage/health. This made it easy to debug, and clear for the player to see what is happening every timestep.

6 Implementation of Design

Here, only a few of the design implementations will be covered as the majority are self explanatory when viewing the code, and all class/method description are shown in the JavaDoc (found in project folder).

6.1 Bank Account

A Bank Account class was created to handle all financial concerns. The checkAccountAndBuy() method looks particular messy, as it has a lot of output questions and input scanning, and handles the whole Tower purchasing procedure (including the placing of said Towers). The balance for the account object created constantly changes throughout the game, and can be retrieved using the getBalance() method.

6.2 Attacking Process

In the advance() method of the Game class, all the towers are cycled through, checking if any are able to fire (calling willFire()). For every Tower that is loaded, the closestEnemy() method is used. It returns the closest enemy in the Euclidean plane by calculating the enemies in the closest x-position (that have not passed the Tower), and if there are multiple, then it returns the first in the List (closest in the y-axis).

6.3 Display Game

There were two main classes used to display the game. The first is named Debug, and was initially used during the testing/debugging stage. It was kept in the game display as it produces useful stats (found by calling the toString() methods).

The ShowGame class was used for the more visual side of things. It may appear long, but was necessary for printing the correct letter and info for each given Enemy/Tower. It utilised for loops to print the walls, and the correct number of spaces before each object to ensure it was placed at the correct position. See **figure 12** for a legend of the visual game display.

Legend	
Game Symbol	Meaning
W	Wall
r	Rat
e	Elephant
o	Orc
u	Uruk Hai
S	Slingshot
C	Catapult
D	Dwarf Axeman
E	Elf Bowman

Figure 12 - Game display legend

7 Testing Procedure

Testing was simple once the UI had been implemented. Viewing the stats step-by-step mad clear any game operations that did not follow the rules, acting undesirably, possibly crashing the game.

Various test cases were performed to check of all possible scenarios, ensuring the game executed smoothly and correctly no-matter the situation.

For retrieving the stats, the **toString()** method was overridden in the appropriate classes, to return the desired information. The classes this implemented in were the two Super-classes (Enemy/Tower), as all the objects that would use this feature inherited from these classes.

8 Proving the Correctness of the Game

8.1 Attempted Firing when Enemy is in and out of Range

Shown below is a Slingshot in position 50. As its range is only 40, it is unable to hit Enemies in the first 10 positions. This is proven by the health not going down when out of this range (shown in **figure 13**). To prove that it does work when in range, view the HP decrease in **figure 14**. Note that the Enemies move position first, before Towers attempt to strike.

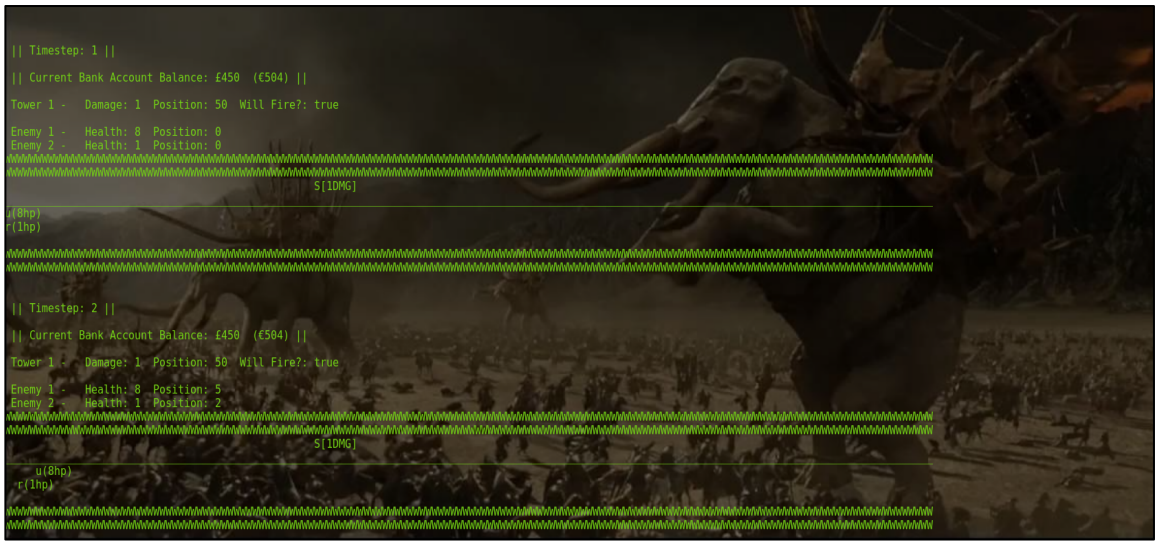


Figure 13 - Out of range example

Figure 14 - In range example

8.2 Checking if Tower is Loaded and Hitting Closest

This example demonstrates that the Tower only fires when willFire() returns true. It also demonstrates the choice of which Enemy to target, by eradicating the closest enemy (an Orc). This can be seen in **figure 15**.

Figure 15 - Closest Enemy attacked from loaded Catapult

8.3 Weapon Not Loaded, Death, and Earnings

Shown in **figure 16** below, when the "Will Fire" was false, the Catapult was unable to attack. However, the Slingshot was able to kill an Uruk Hai that was on 1 HP. It can be seen that by the next timestep, its body was removed from the battlefield, and the player's bank account balance increased by £100. This death/earning process can also be seen in **figure 15**.

Figure 16 - Non-fire, death, and earnings

8.4 Lost the Game

Figure 17 demonstrates the end of the game's execution, which is activated when an Enemy (in this case 2 Uruk Hais) cross over the border into the Rivendell, the defended territory.

(Please play the game to try and find the output result when the game is won!)

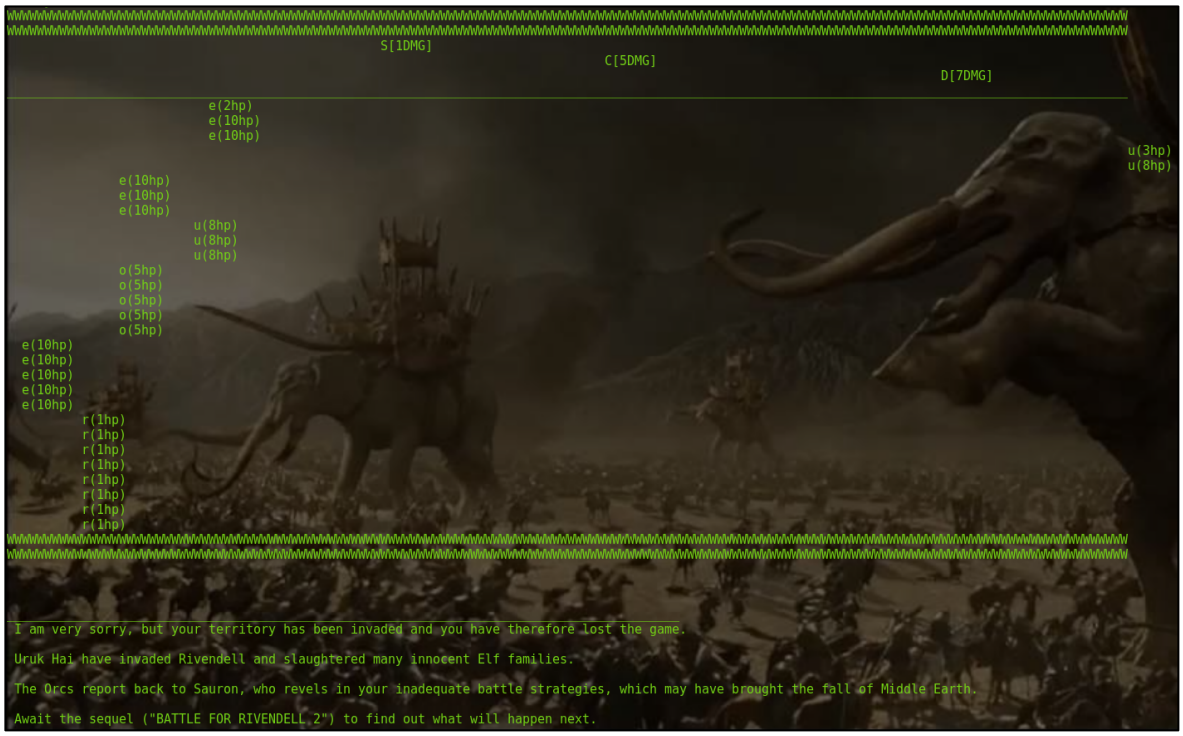


Figure 17 - Lost Game screen

9 Running Instructions

To run the game, simply navigate into the src folder by following the path:

…/CS5001-p2-oop/src

From here, execute the following command to look into the package, and then run the class with the main method in it (in my case, this is the 'Game' class):

java towerdefence.Game

This should work, as the class files have been included, but if there are any problems, try compiling all the classes again (javac), and then run it using the previous command.

10 References (URLs) for Pictures

[1] https://vignette.wikia.nocookie.net/saganomringen/images/8/82/Rivendell_-_The_Hobbit.png/revision/latest?cb=20131021205838

[2] http://www.thelandofshadow.com/the-wargs-in-the-hobbit-vs-the-wargs-in-the-lord-of-the-rings-films-by-peter-jackson/

[3] https://vignette.wikia.nocookie.net/lotr/images/9/95/180px-Oliphaunt.jpg/revision/latest?cb=20140329114404

[4] http://lotr.wikia.com/wiki/Orcs

[5] https://i.ytimg.com/vi/tgXPRxmHk6Q/maxresdefault.jpg

[6] http://www.alloutdoor.com/wp-content/uploads/2016/08/slingshot-cannon-660x380.jpg

[7] https://vignette1.wikia.nocookie.net/edain-mod/images/4/44/Troll_Catapult-0.jpg/revision/latest?cb=20160817212451

[8] https://vignette1.wikia.nocookie.net/the-hobbit-and-the-lord-of-the-rings/images/e/ed/Dwarf.png/revision/latest?cb=20121229162907

[9] http://media.indiedb.com/images/members/1/762/761504/Mirkwood_Elite_Archer.png