Qt Tower Defence Practical

REII313

by:
Dewald Krynauw 26013835

North-West University Potchefstroom Campus

Lecturer: Mr. A Alberts

June 4, 2017

Abstract

Kaasbroodjie

Contents

List of Figures

Abbreviations

GUI	Graphical User Interface
UDP	User Datagram Protocol
IP	Internet Protocol
TD	Tower Defence
OOP	Object Oriented Programming

1 Introduction

Tower Defence games has been a popular genre of gaming ever since graphics based strategy games became well known. The principle of how they generally work is, to stop the enemy minions from entering the player's base/end-point. This is done by placing different sorts of towers in a strategic manner to stop them.

For this assignment, software has to be written to create a Tower Defence game. Our knowledge on Object Oriented Programming (OOP) will be tested in the developing environment of Qt C++.

The game has to adhere to a few design specifications in order to be sufficient.

- Have a grid-based tower placement.
- Apply A-star path finding for enemies to follow.
- Single-player playability.
- Networking and multi-player playability.

In the following sections the process on how the TD game was designed and implemented is carefully examined. Further elaborations on the output and shortcomings are made below.

2 Layout and Design

In this section the overall GUI is analysed. A quick summary of what each part does in given, but will be discussed in more detail in further sections.

2.1 Main Menu

When the game is first run, the user will visualize the main menu output in Figure 2.1 with four buttons: Play, Host, Join and Quit.

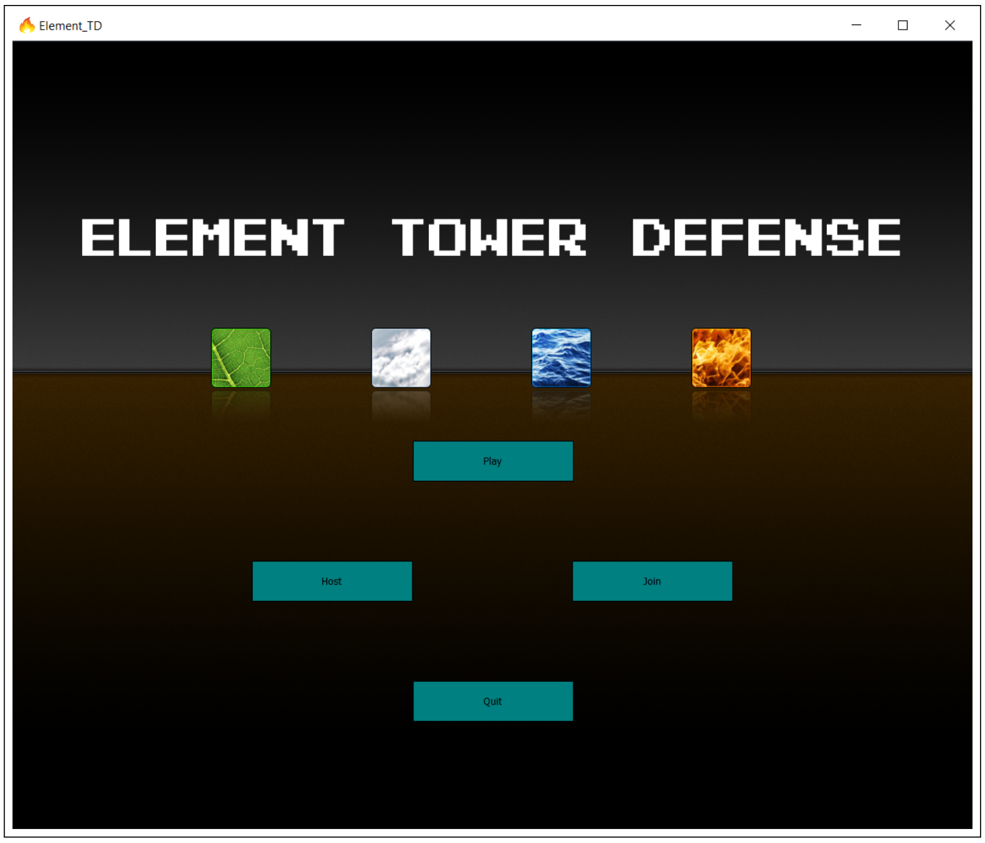


Figure 2.1: Main Menu

Play Button
Starts a single player game mode on the local loop-back address.

Host Button
Goes into a waiting loop until a joiner sends an acknowledgement over the network. When

the acknowledgement is received, a new game starts.

Join
A input dialogue box appears. The user can the input the host's address. After the OK button is pressed the user send an acknowledgement to the host and starts a new game.

Quit
This simply quits the program.

2.2 Game GUI

When a new game is created the user will see the output generate in Figure 2.2. The user can then see the grid on which he can place the desired towers. The towers are located in the bottom right panel. Player statistics are displayed in the top right panel.

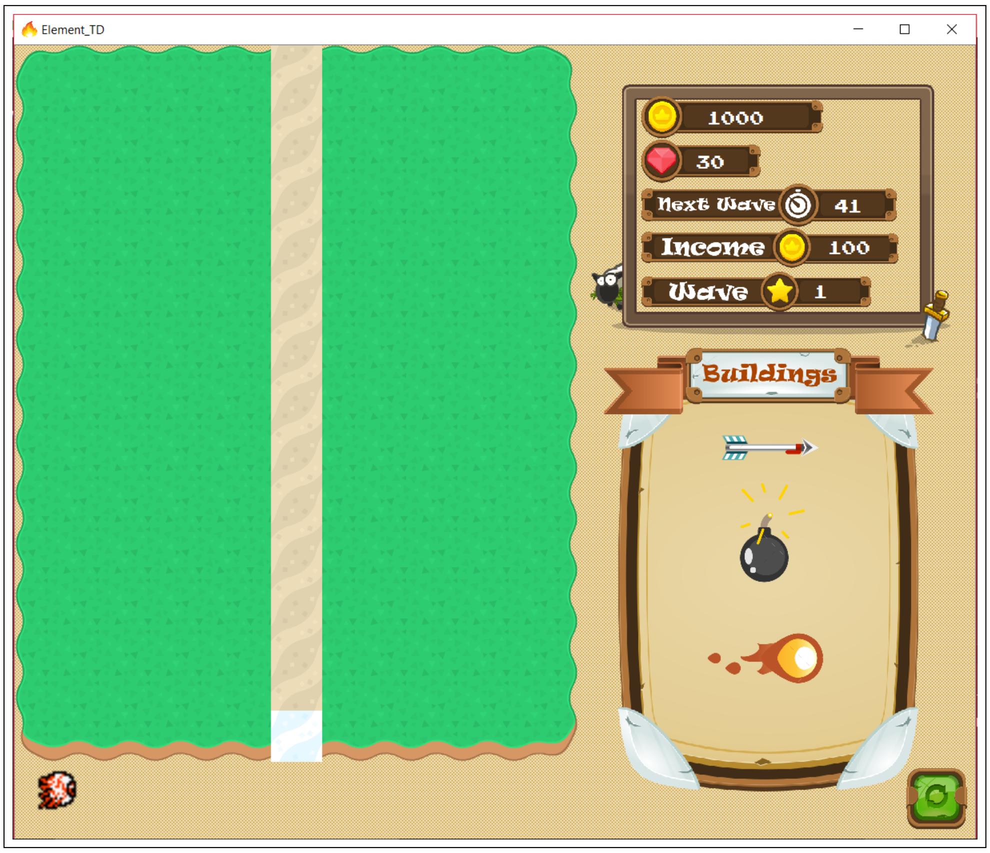


Figure 2.2: Game GUI

Eye Icon
In the bottom left of the GUI, the user will see an "eyeball icon". When the user clicks on it, it spawns an enemy at a cost, but increases the income gained when each wave starts.

Restart Icon
In the bottom right of the GUI, there is a "Restart Icon". When pressed the user will return to the main menu.

Building Icons
When the user clicks on one of the building icons, an instance of the tower should be created. When the user then clicks on the grid the tower should be placed.

3 Logical flow of Program

To better understand the flow of the program, each of the major phases are broken down in the following section.

3.1 Game Mode Selection

When the game is first started, the main menu will display. The user must the choose which game mode to play.

- Single Player
- Host a game
- Join a game

Single Player
This creates a local loop back on the player's own IP using the UDP protocol. Every time the player presses on the "Eye Icon" it actually sends the enemy over the network, but to the player's own address. The instance created in single and multi player mode is the same, the only difference is the network socket.

Host a game
This creates a waiting loop, that polls for whenever an "ACK" is received over the network. When the "ACK" is received, it binds the UDP socket to the render address from which the "ACK" came. A new instance of the game is created.

Join a game
This opens an input box for the user to enter the Host's IP. After "OK" is pressed an "ACK" is sent over the network to the Host. A new instance of the game is then started.

3.2 Phase 1: Upkeep and Building Phase

When the game instance in created; the player is awarded a certain amount of gold, lives and income with a new clean map with selectable buildings icons.

When the player clicks on the building icon, the selected building is created at the mouse pointer location. The building follows the mouse pointer where ever it moves.

The player can then click on the map to snap the tower to the grid and place it. After being placed the specific amount of gold for the tower is deducted from the gold.

Multiple towers can be placed further, until all the gold has been expended or the wave timer runs out signalling the next incoming wave.

3.3 Phase 2: Wave Phase

At the start of each wave the following things occur. The player's gold is increased by the current income amount and the respective amount of enemies are spawned.

Income
Income is the amount of gold the player will receive at each wave level. Income can be increased by sending enemies over the network, whether it be single or multi player. The user pays a large amount to send an enemy but gains a small increase in income.

Wave Level
Each time after the wave timer counts down to zero, a new wave of enemies are spawned. At level 1, only one enemy is spawned. At level 2, two enemies are spawned and so on. With each increasing wave the enemies' health and number enlarges.

Enemies
When an enemy is spawned is starts at the top of the map and starts moving in the shortest path available to the bottom portal. It does the path finding by making use of the A-star algorithm to calculate the shortest path.

3.4 Phase 3: Post Wave Phase

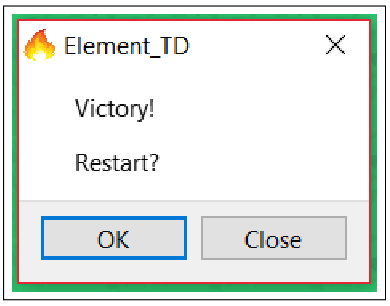
After the wave of enemies have been spawned and start following the set out path. The towers will start shooting at the enemies as soon as they are in range.
Each type of tower will do different amounts of damage to the passing enemies. When the enemy health drops to zero the enemy is killed and gold is awarded to the player.

If the player should fail to kill and enemy and passes through the portal; the player will lose a life. If the player's lives reaches zero the player loses the game.

3.5 Phase 4: End Game

Single player
After the all of the player's life have been depleted, the game will stop and display a message box asking whether or not the user wants to restart or exit the programs.

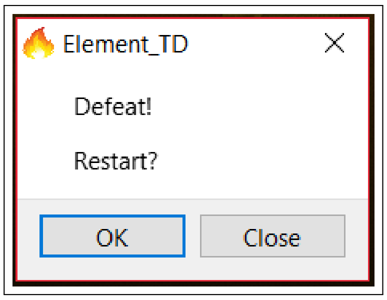
Multi player
The first player to reach zero lives will get a "Defeat" message. The defeated player will send a "Game Over" over the network. When the command is received the victor player will receive a "Victory" message. Both of the users can then decide whether or not to retry the match.


Element_TD

Victory!

Restart?

Figure 3.1: Victory Message Box


Element_TD

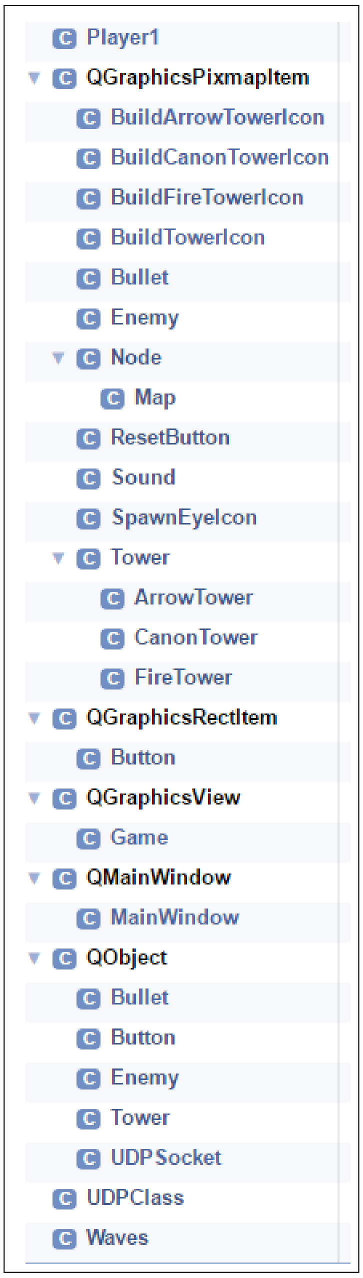
Defeat!

Restart?

Figure 3.2: Defeat Message Box

4 Class Hierarchy

This is the hierarchy of the classes generated by the Doxygen software.



```
Player1
▼ QGraphicsPixmapItem
    BuildArrowTowerIcon
    BuildCanonTowerIcon
    BuildFireTowerIcon
    BuildTowerIcon
    Bullet
    Enemy
  ▼ Node
      Map
    ResetButton
    Sound
    SpawnEyeIcon
  ▼ Tower
      ArrowTower
      CanonTower
      FireTower
▼ QGraphicsRectItem
    Button
▼ QGraphicsView
    Game
▼ QMainWindow
    MainWindow
▼ QObject
    Bullet
    Button
    Enemy
    Tower
    UDPSocket
UDPClass
Waves
```

Figure 4.1: Class Hierarchy

5 The Game Class

The Game class serves as the major control class. Here most of the logical flow of the game is managed. The game object is created at the start of the game. It creates the QGraphicsScene on which all the other objects are created.

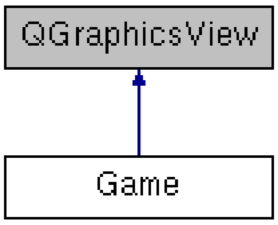


Figure 5.1: Game Class Diagram

5.1 Design

The game class will inherit from the QGraphicView public that provides a widget for displaying the contents of a QGraphicsScene. The QGraphicsScene also provides functionality that lets you efficiently determine both the location of items, and for determining what items are visible within an arbitrary area on the scene. The scene will function significantly better than the use of labels by making use of the QGrahicsItems objects.

It will have to be a Q_OBJECT in order to handle signals and slots.

The Game class must be able to do the following,

- Display Main Menu
- Start Game
- Handle Tower Placements
- Create Enemies
- Print Map with A-star
- Player Stats
- Create Network Socket

5.2 Implementation

Display Main Menu
As soon as the program starts a form with a background and 4 self written button classes. Each being connected to its own slot. Each button is a type of QGraphicsPixmapItem with

text as input parameter. These buttons emit a signal when clicked.
The start button run the startGame() function.
The host button runs the waitConnection() function.
the join button runs the join() function.

Start Game

Firstly the game clears any previous structures leftover from the previous game modes.
It then creates a new map from the Map Class.
Creates a new player class.
Creates a new spawn timer.
Adds a stats frame with gold, lives, income, wave timer and wave lives.
Adds reset and sound mute button classes to scene.

Handle Tower Placements

The tower placement is one of the most complex processes in the game. There are two major mouse events that controls the placement, ”mouse move” and ”mouse press” events. Firstly the mouse move event.

After one of the building icons have been clicked, it sets a cursor and goes into building mode. The cursor is a pixmap that then follows the mouse pointer wherever it goes, until placed.

If the ”mouse click” event is activated, it will check the if the game is in building mode, if so, it will clear the old path, get the closest node to which the player clicked, and snap the tower to the grid. Otherwise it does nothings.

For the tower to be a valid placement it has to satisfy **all** of the following constraints.

- Not be an obstruction tile
- Have more than 0 gold
- Have a valid A-star path to follow

If these constraints are not satisfied cancel the click operation.

Create Enemies

With each wave the game will create enemies. This function simply connects a 1 second timer with the number of enemies that need to be spawned.

The spawn enemy function is then called each time the timer's slot triggers. The spawn enemy function then create a new enemy with points to follow as a parameter. The enemy's health is increased each wave. The enemy is added to the list of enemies.

Print Map with A-star

This function checks each of the map tile types, it then prints the map with the new tiles each every time the A-Star algorithm is run.

Player Stats
At the start of the game the player statistics are determined by player class when initialized. This just initializes the default values to all the stats when the game starts.

Create Network Socket
In the game constructor UDP socket is created with the default host address and the local loop back address.

6 Network Protocol Class

This class is in charge of handling the UDP socket it creates at start-up. It describes the entire protocol used in the program and how communication is established.

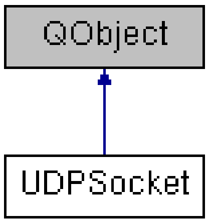


Figure 6.1: UDPSocket Class Diagram

When the class in constructed, a new QUdpSocket in created. The socket is the connected to the readRead() signal and a read datagram slot. While the socket has pending datagrams to read, it processes the datagram.

If the datagram is the word ”spawn” it spawns a single enemy.
If the datagram is ”ACK” it sets the host address to the sender address and flags that there is a game instance.
If the datagram is ”GO” signalling that the other player has a ”GAME OVER” after which the game class victory function is run.

7 Map Class

The Map class provides the grid and tiles displayed, but also all the path finding functionality. It determines whether there is a valid path for enemies to follow. Whether a certain node is and obstruction/occupied.

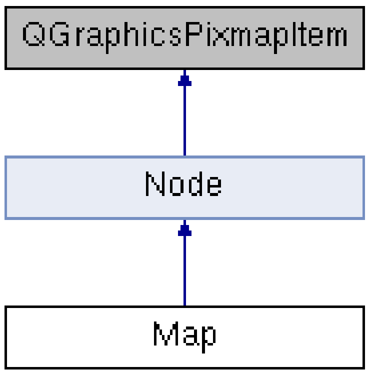


Figure 7.1: Map Class Diagram

When a new map is constructed, it assigns all the map nodes to default values. It also assign the starting and end nodes.

This class is in charge of handling, all of the A-star path finding functions such as,

- Calculating the H value
- Calculating the Neighbouring values
- Determining the smallest F value
- Getting a node
- Getting the location of all the node points

7.1 Nodes

The node class is simply a QGraphicsPixmapItem that contains all the values to do the path finding.

Each node has a,

- Tile type
- Cost
- X and Y
- F, G and H value
- Parent Node

- Path boolean

The different tile types of the node can be seen in the figures below.

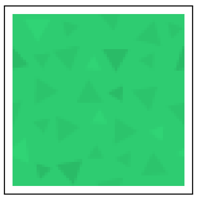


Figure 7.2: Grass Tile

Figure 7.3: Path Tile

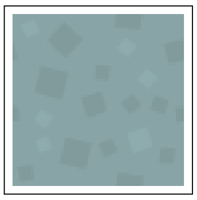


Figure 7.4: Obstruction Tile

8 Tower Class

This class is the base class of the different types of classes. Other tower classes such as the fire tower, inherits from the Tower Class. This is where polymorphism is implemented. This class determines whether or not an enemy is within range and acquires the nearest target. Then calls the virtual ”fire” function, to make the derived tower class shoot.

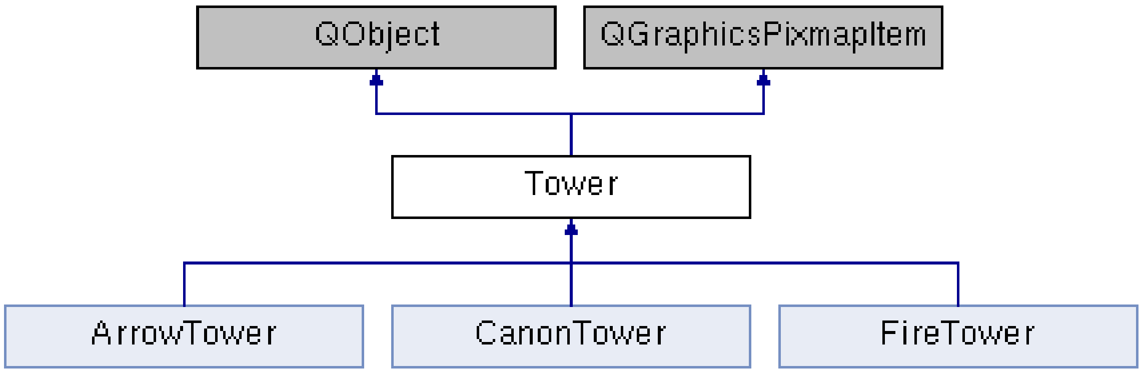


Figure 8.1: Tower Class Diagram

Firstly an attack area around the tower is defined and get a list of all enemies that collide with attack area, find the closest one and set it’s position as the attack destination. It creates a bullet as seen in Figure 8.2 and a line between the tower and attack destination for the bullet to follow and fires the bullet.

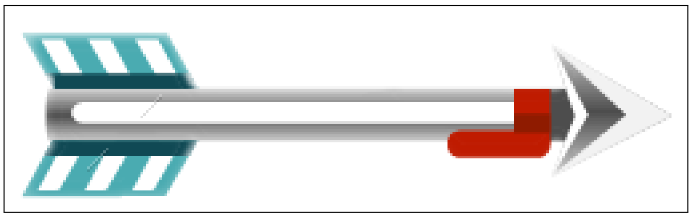


Figure 8.2: Arrow Tower Bullet

The default arrow tower can be visualised in Figure 8.3 below as well as the upgraded level 2 tower. After the default tower has been placed and double-clicked, the tower is upgraded; increasing its damage and look. But the player has to pay a small upgrade cost.

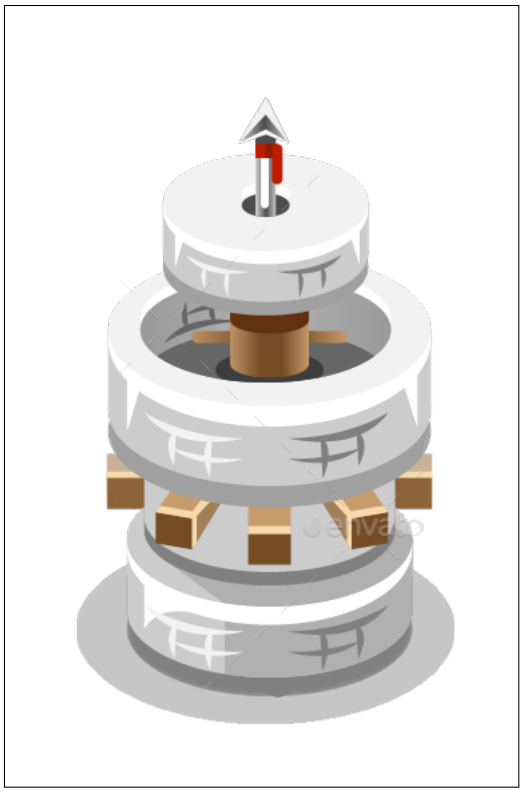


Figure 8.3: Arrow Tower (Level 1)

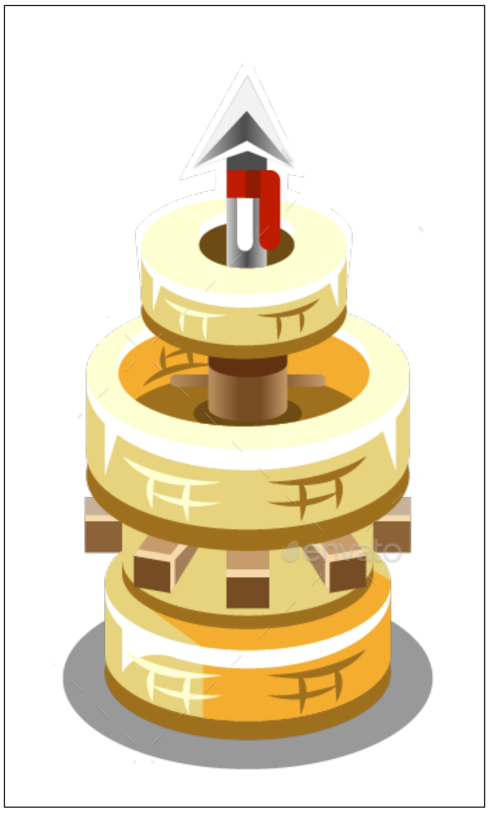


Figure 8.4: Arrow Tower (Level 2)

9 Enemy Class

This is quite a simple class, but play a crucial part in the game. It creates an enemy object each time the next incoming wave arrives, or if someone sends an enemy over the network. The enemy then traverses the shortest path to the final node generated by the A-star algorithm. Each time the enemy is shot, it takes that damage from its health. If it reaches zero it is removed from the game. If it makes if to the end, the player's health is deducted from each enemy that passes.

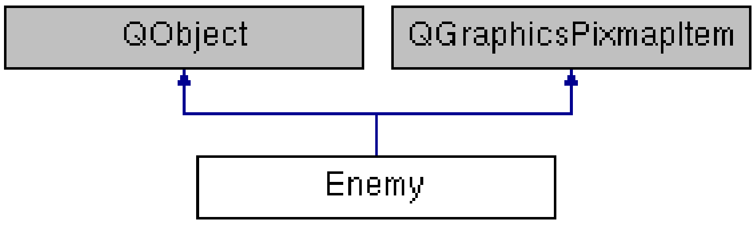


Figure 9.1: Enemy Class Diagram

Figure 9.2: Eyeball Enemy

10 Analysis of Results

At the start of the game a wave timer starts counting down until the next wave arrives. In this time the player has time to place towers in his desired manner. This process of building towers in order to survive is repeated at each wave level. Like in Figure 10.1 the game starts of simple; with a small amount of enemies and not too many towers.

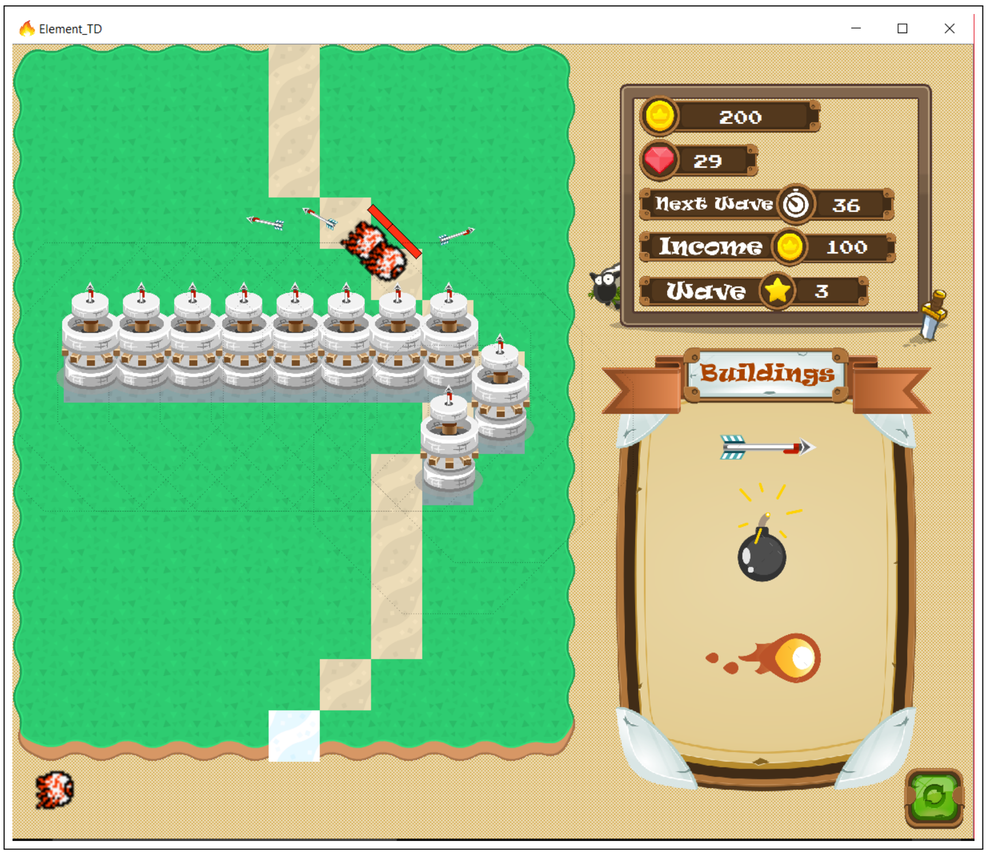


Figure 10.1: Starting Round Output

After a while (usually after the 10th wave) the game starts to experience lag due to the amount of enemies and bullets firing. A common solution would be to use threading as well as less graphics intense images. This can be seen in Figure 10.2.

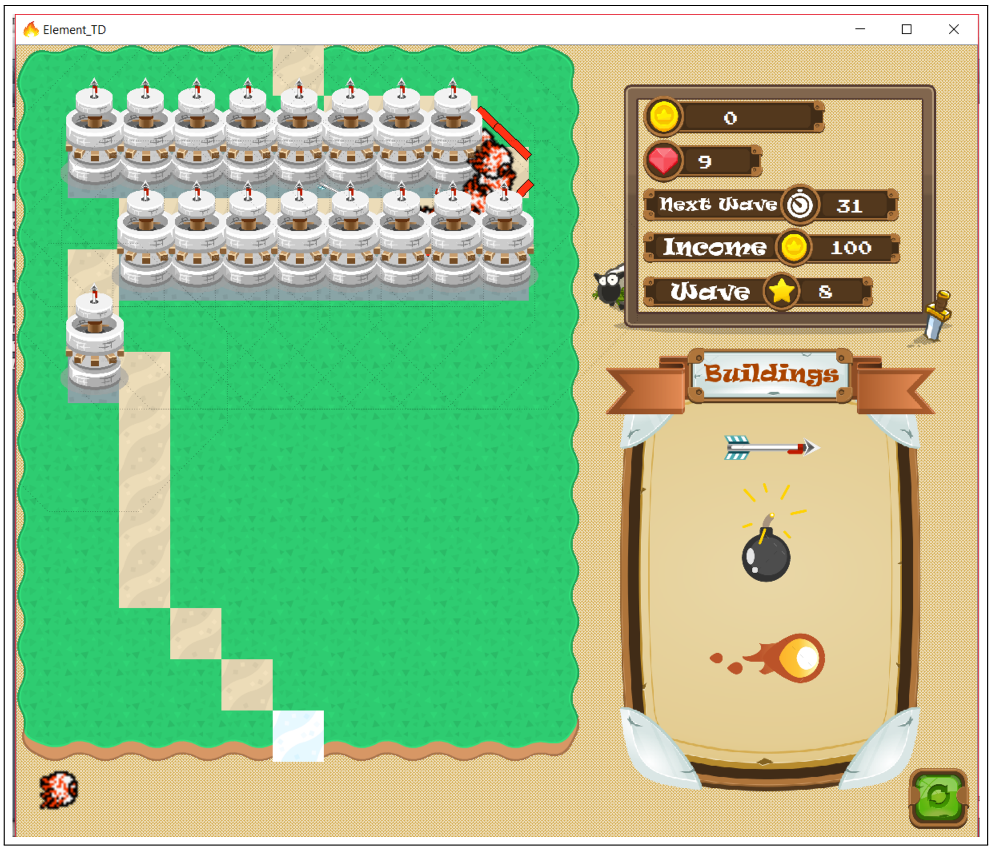
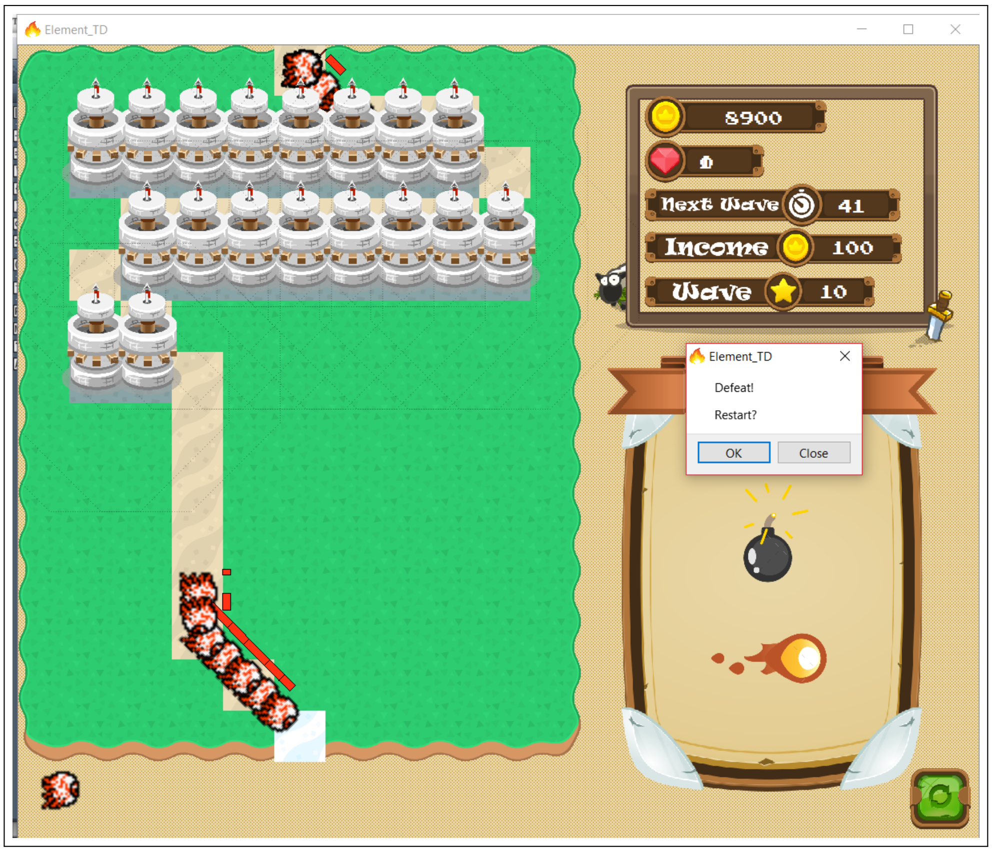


Figure 10.2: Later Rounds Output

After the player has been defeated, a message block will appear and ask if the player wants to restart like in Figure 10.3.



Defeat!

Restart?

Figure 10.3: Defeat Dialogue

11 Conclusion

This is a fun and exciting game that challenged the programmer's core programming as well as Object Oriented Programming skills. A functional game can be written in the Qt environment, although should the task be to write a game, the use of a game engine, such as Unreal Engine would be much more logical. But for the purposes of this practical, writing a working Tower Defence game is possible. There are a few shortcomings in the development of this game, should there be a chance to do everything over again, the following things could be improved upon.

Balancing the game proved the most challenging underdog of the game. A well balanced game separates a good game from a great one. This is one of the trickiest to do and the least attention was paid to it. There are times in the beginning where the game seems ridiculously difficult as to where the late game is too easy. The health of the enemies were then set to grow with each wave level, so that it becomes incrementally more difficult. This did change the dynamic of the game a bit, but not to a point of pure satisfaction.

Another solution could have been to have less monsters and just increase their health. Then there also wouldn't be a problem with lag and overloads of money in the late game.

The Network Protocol used in this game was practical to the extent of the game, but could use a lot of improvements, especially in the area of error handling and security. For a game such as this, there is no real need for strong security or encryption protocols, this goes with the use of UDP. Although error handling, syncing and good communication makes a game much more stable over a network. A crucial part perhaps where this game was lacking is that there are never conformation whether or not someone received the massage that was sent and whether it is okay to continue with the current command.

This could easily be done by replying to a certain command that will make the game approve the datagram and respond to it accordingly.

Lag is another major problem in this game. In the early game up until level 10, the game runs very smoothly, but when the amount of enemies and bullets become too many; the game starts lagging out.

A way this was compensated with was to shrink the sizes of the bullet so that they are only a few pixels large. Less enemies were added each level. The pixmap quality was lowered. The game was also run in release mode. This significantly improved the lag, but was still present.

It has to be noted that this wasn't a memory leak, just too many objects were created and not cleared fast enough. Another way would have been to place the bullets or enemies in threads.

Parenting was one of the traits that were slacking in this game. There was room left for correct parenting, but did not affect the game's performance or functionality, but in order to avoid any memory leaks the parents of the object of the game could have been assigned to the game class, so that they are properly de-constructed when the can closes. This only caused some inconvenience at build time, because there were many unassigned parents. This problem could be easily fixed.

Lastly a word on ***Cohesion***. Just because a game is functional doesn't always mean that it is well programmed. Due to the nature of OOP, there are bound to be redundant library includes or overlapping, but in general after a few clean-ups the program would be written really well in terms of functional cohesion. The functions are class specific with appropriate parameters. Understanding the code and adding features even after a long period of time remains easy and understandable.