REII 313 Practical

C++ Game Design

by:
J Koekemoer 26035170

North-West University Potchefstroom Campus

Lecturer: Prof. A Alberts

June 4, 2017

Contents

List of Figures

1 Abstract

This practical report concerns the design and creation of a C++ tower defence game in the Qt programming environment. The resulting game included the following features: original sprites, a user-friendly GUI, tower placement, an income system, a life counter, timed enemy waves of increasing strength, enemy pathfinding, animation, target acquisition, bullet creation, and hidden lore. Although Scroll Keepers is largely playable, it has room for improvement. However, there are definite plans to upgrade it in the near future.

2 Introduction

In this practical report, it is discussed how the objective of this practical was approached. The objective was to create a functional tower defence game using the C++ programming language. This was, ultimately, in order to test if the programmer was sufficiently adept in several programming aspects. The game should be programmed in Qt, the coding environment which was used for the duration of the REII 313 module. Several requirements should be met for the game to be considered functional.

2.1 Objectives

- The game must have a graphical user interface (GUI) which uses widgets and events, as well as provide for input.
- The game should have logical controls, dynamics, and flow.
- the game should implement collision detection, path finding, and somewhat intelligent non-player characters (NPCs).
- There should be a network component to the game which allows for a synchronised multiplayer mode, and must use a predetermined communication protocol, such as transmission control protocol (TCP).
- The game must be fun to play.

3 Game Overview

3.1 Game Description

The aim of the game Scroll Keepers is to place dragons in a manner that prevents any shadow enemies from reaching the shelf full of ancient scrolls. Dragons are purchased using the crystal resource. The enemies appear in waves, and attempt to reach the scrolls.
The player wins if there are still ancient scrolls left once all of the enemy waves have completed. However, the game is lost if the player allows all of the scrolls that you are trying to protect to be stolen

3.2 Characters

3.2.1 Dragons

The game features dragons as the defenders of the ancient scrolls, and the name "Scroll Keepers" originates from this fact. Their designs can be seen in Figure 3.1. The dragons live in the library, and require crystal energy to hatch. Once they hatch, the dragons fire bullets at the shadow enemies.

The electric dragon is quick to hatch, as it has a lot of energy of its own. It fires weak balls of lightning at enemies at a standard rate;
The plant dragon takes a little longer to hatch than the electric dragon. It summons the power of nature to scatter enchanted leaves on its opponent, dealing decent damage.
The fire dragon needs a fairly long time to build up enough heat required to hatch. However, once hatched, it shows off its impatient nature by shooting fireballs at its foes at an impressive rate. Sadly, this sacrifices its accuracy.
The ice dragon is the strongest of all dragons, but hatches with glacial pace. Once it hatches, it casts powerful ice shards at the shadow enemies.

Figure 3.1: The dragon sprites, along with their respective bullets and eggs, for the game Scroll Keepers

The Mad King is also a dragon, but he is the mysterious villain in the game. He can control the shadows to do his bidding, and sends them to bring back scrolls for some unknown, yet probably nefarious, purpose. A partially hidden portrait of him can be seen in the library, shown in Figure 3.2.

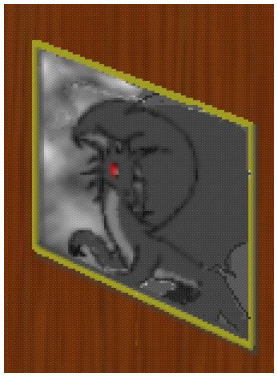

Figure 3.2: The obscured portrait of the Mad King

3.2.2 Shadow Enemies

Shadow enemies are the servants of the Mad King, and are charged with stealing ancient scrolls for him They can be seen in Figure 3.3. The shadow enemies spawn periodically from a sinister portal in the library window. They will find a path to the shelves containing the library's ancient scrolls. Once they reach the scrolls, they will take as many as they can carry and warp back to the shadow realm. They drop crystals when defeated.
The bird shadow is an arbitrary enemy minion with low health. Furthermore, it can only steal one scroll. This makes them the perfect practice enemies.
The hound shadow is a shadow enemy of decent strength, having a larger health than that of its bird shadow counterpart. The hound shadow is capable of stealing two scrolls from the library.
The shadow lord is a deer-like shadow enemy which has a lot of health, and is capable of stealing five scrolls. It is the strongest enemy in the game.

Figure 3.3: The shadow enemy sprite set

3.3 Resources

Scroll Keepers has two resources, namely crystals and scrolls. They influence the player's available choices and indicate how well they are playing.

3.3.1 Scrolls

Scrolls are the resources which the player is trying to defend. They can be thought of as "lives", as when there are none left, the game has been lost. There are always fifty scrolls in the library at the beginning of each game. Figure 3.4 shows the scroll icon.

Figure 3.4: Scroll icon

3.3.2 Crystals

Crystals are generated linearly over time, and are also awarded for defeating shadow enemies. Crystals are used to provide dragon eggs the energy they need to hatch. The crystal icon is shown in Figure 3.5. For various reasons, different types of dragons require different quantities of crystals to hatch.

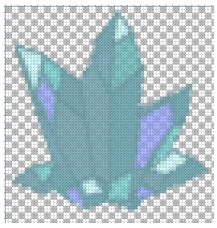

Figure 3.5: Crystal icon

3.4 Environment

The game Scroll Keepers takes place in a historical library inhabited by dragons. There are bookshelves up to the ceilings, along with smaller shelves in the room. These act as the obstacles for the shadow enemies in the game.
There are towering crystals surrounding the shelves holding the ancient scrolls. Large windows reflecting Gothic architecture are the primary light source for the library, but they are currently being used as an entrance for the shadow enemies.
The floor is tiled in some sections, and carpeted in others. There is also a stone section in front of a sturdy wooden door, which, presumably, leads to either a dungeon or a broom cupboard. The various tiles used can be seen in Figure 3.6 below.

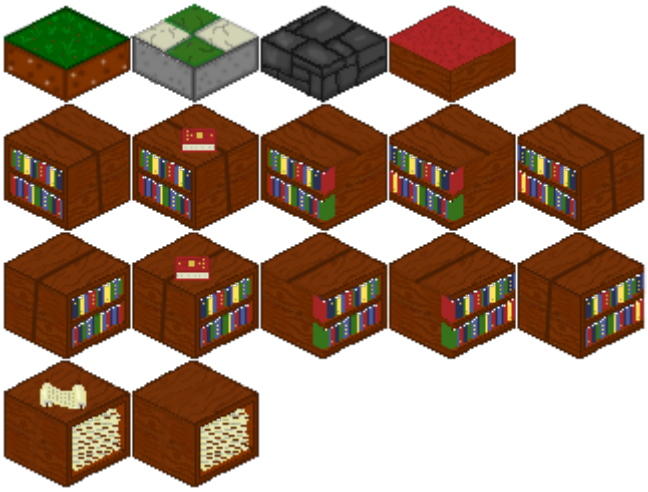

Figure 3.6: Scroll Keeper's tile set

3.5 GUI and Controls

The player controls the choice and placement of the dragons by clicking on their respective egg buttons on the GUI. The player clicks their egg of choice, and clicks again to place the egg in a chosen position on the map.
The available resources are also displayed on the GUI, and are updated each time they change. For an egg to be placed, the player must have (and pay) sufficient resources to hatch the specific type of egg.
The egg buttons and resource displays are grouped together on a heads-up display (HUD) on the bottom of the screen. The HUD, shown in Figure 3.7, resembles a balcony, giving the impression of looking down on the library map.
The game window can be minimised, maximised, and closed by using the standard buttons in the window's upper right corner.

Figure 3.7: Scroll Keeper's HUD showing the egg buttons and resources

4 Game Design

For a game to be successful, it needs to follow a constantly reviewed and adjusted design. Once a concept design has been decided on, its logic needs to be checked. Improvements can then be made upon the design if and when issues are found.
Once a design has been thoroughly examined and deemed fit for use, it must be correctly implemented. The design's implementation can often be carried out in several different ways, and choosing the best methods is crucial to the program's success. The chosen implementations must be compatible with one another if the sections are intended to work together properly.
How different sections of the program work together is important to plan and keep track of. This is where class diagrams are most useful.

4.1 Concept Art

While the decision was still being made on how to approach this practical, several ideas were jotted down on paper to aid with the game planning. These would later become key inspirations for Scroll keeper's graphics and gameplay elements. The original concept art can be seen in Figure 4.1, Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.5.

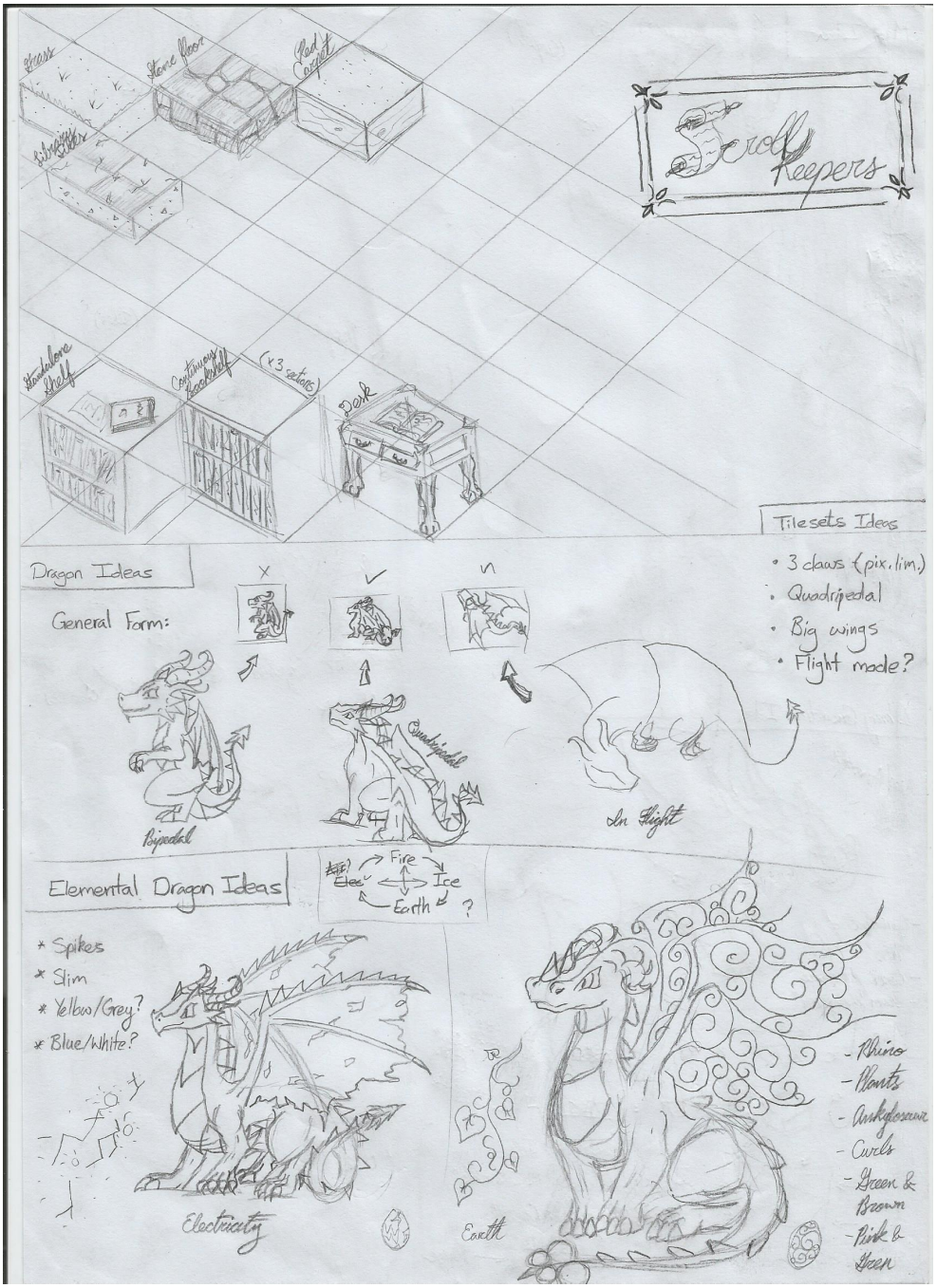

Figure 4.1: Concept art for the various map tiles, as well as an electric and earth dragon (which became the plant dragon)

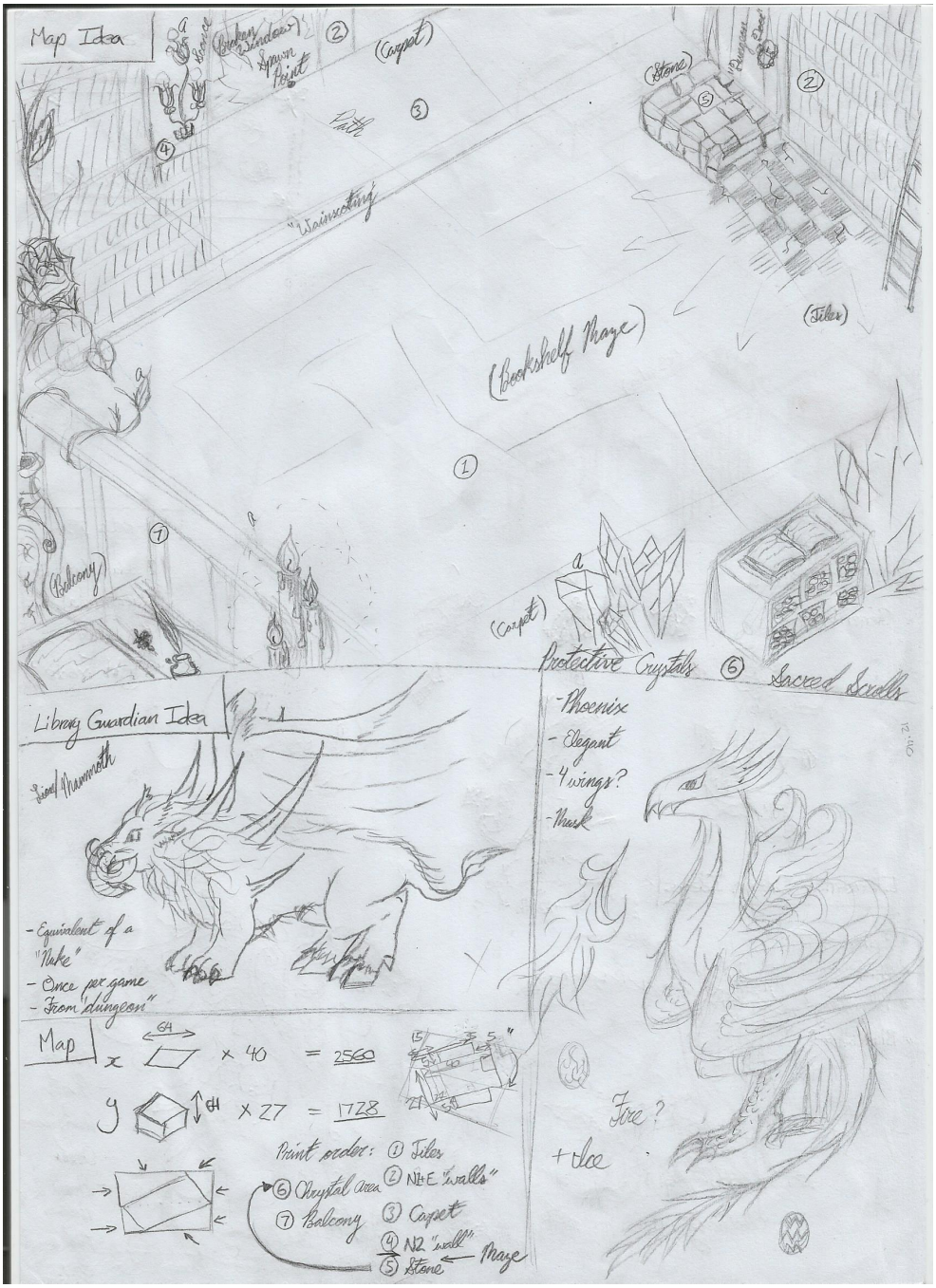

Figure 4.2: The general map concept, and a fire dragon, along with a unit that was not implemented

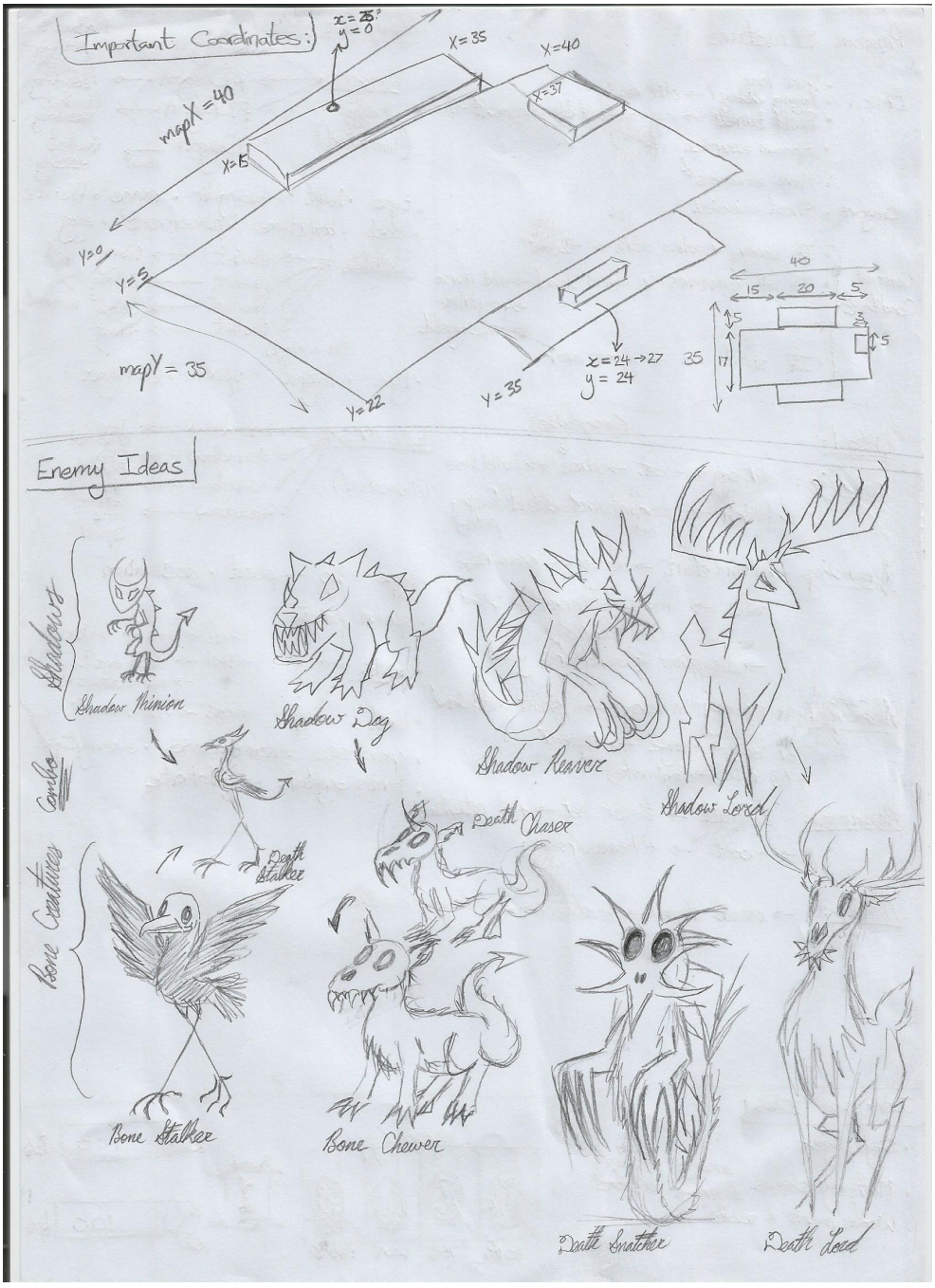

Figure 4.3: A sketch showing map dimensions and possible enemy designs

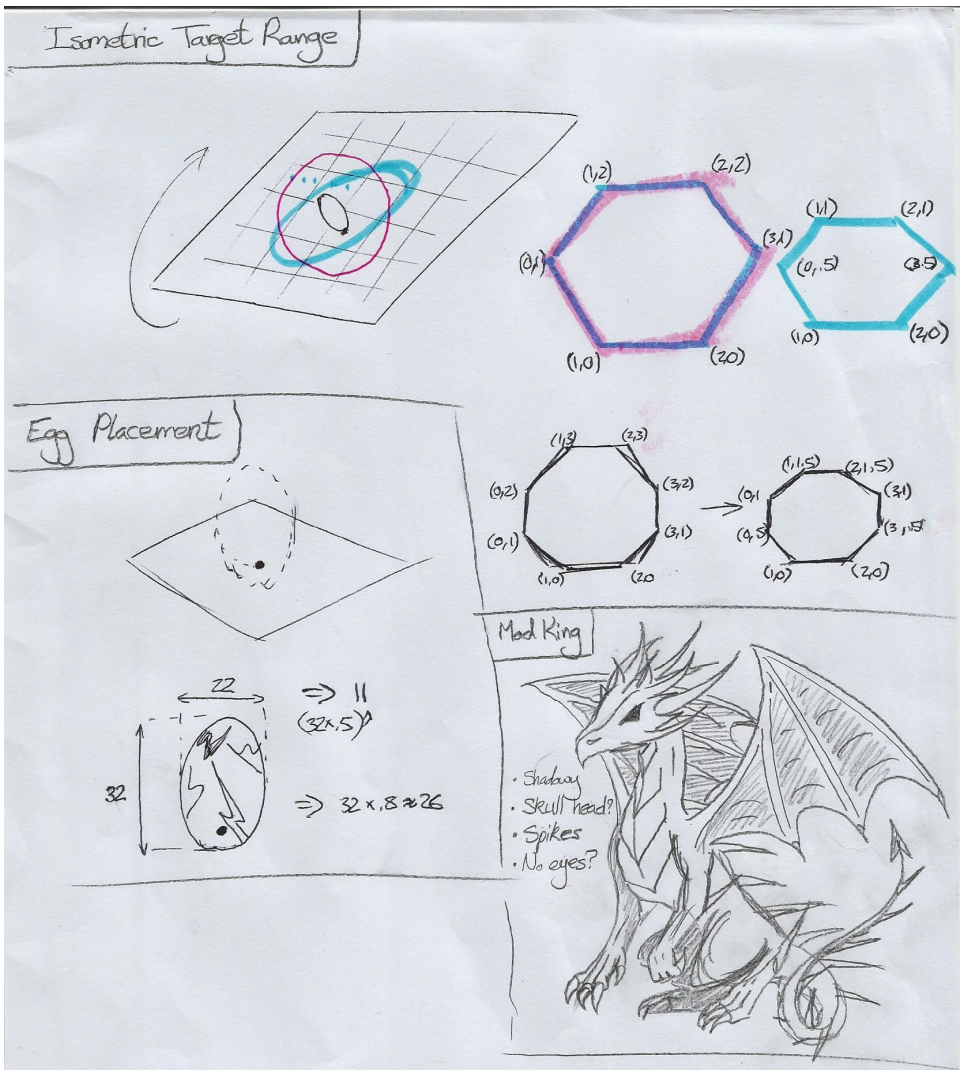

Figure 4.4: A visualisation of the dragon's target range adapted to the isometric view, along with egg placement offsets and the Mad King

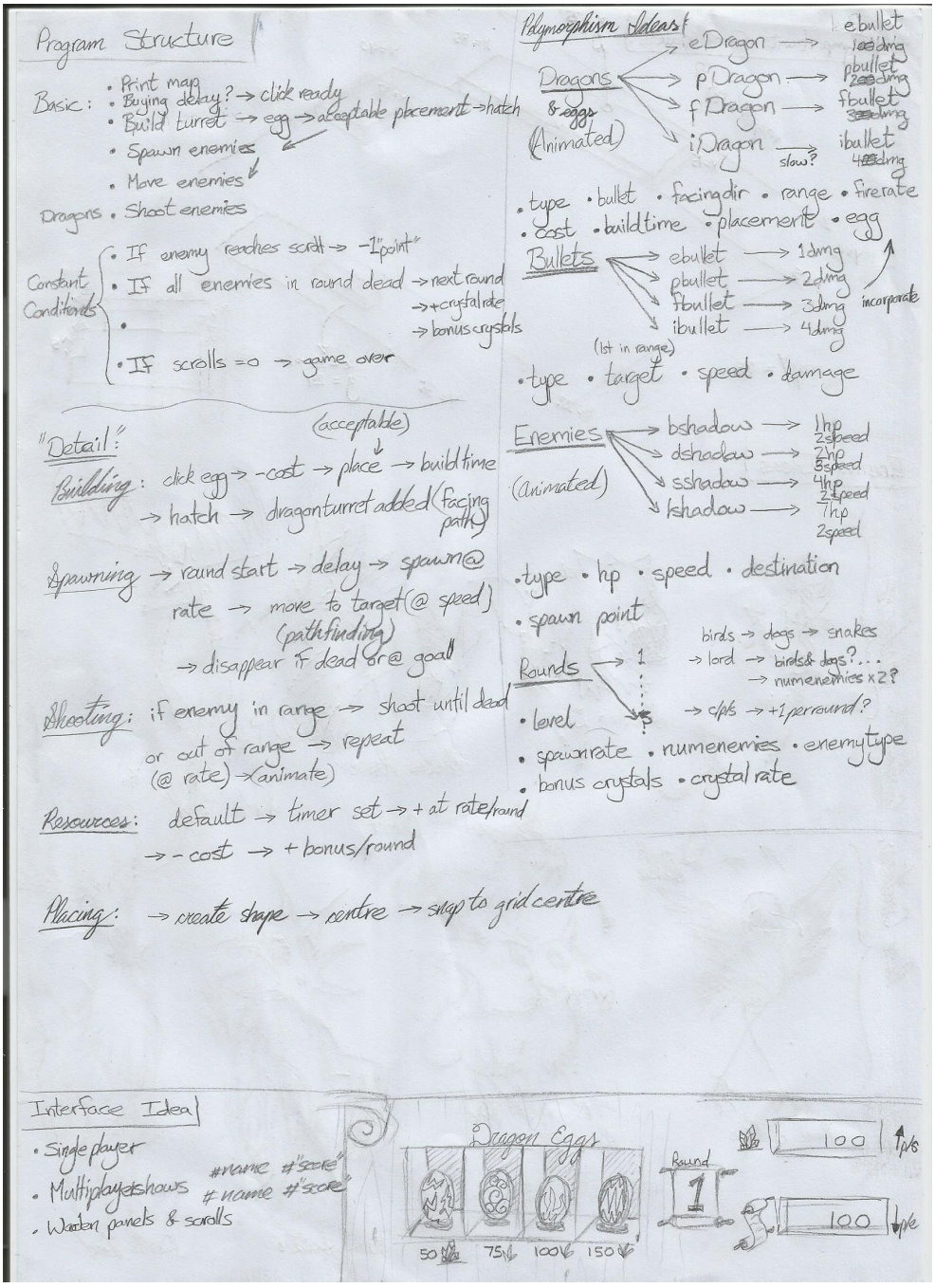

Figure 4.5: An early program logic procedure flow, and a HUD sketch

4.2 Various Program Aspects and Their Implementations

There are several main features in Scroll Keepers which can be identified easily. They will be discussed in the following subsections.

4.2.1 Map Creation

The map in Scroll Keepers is made up of two different sorts of QGraphicsPixmapItems. One sort is the tiles which are printed in two layers from the upper right-hand corner to the lower left-hand one using for loops. Bookshelf tiles are then printed over the map in the second layer in a similar way, but with more complex for loop structures.
The other sort of QGraphicsPixmapItems are the large images which are used to create the library walls and the HUD background. They give the game the cosy atmosphere that a library needs, as well as supporting the general fantasy theme.

4.2.2 Pathfinding

The pathfinding method in this game is used to guide the shadow enemies to the scroll shelves as quickly as possible around obstacles. The bookshelves are the obstacles, and they form a maze around the map.
The A* searching algorithm is implemented in the program, as it is effective, yet simple enough so that the computer system running the game should be able to compute the paths at an acceptable rate. The enemies then get passed the vector containing the points they need to follow. To smooth the enemy's movement, they keep moving only a fraction of the distance to the next point until it is reached.

4.2.3 Animation

Both the shadow enemy's and dragon's sprites are animated. The enemy sprite animates each time it moves, to keep its steps in sync with its movement. The dragon animates according to its own animation timer, as it does not move around the map.
The animation itself is achieved by switching between sprites each time the animation function is called. This is done using a switch/case statement, and is based on the previous sprite used. Each enemy and dragon has an initial sprite with which to begin this process.

4.2.4 Building

Dragons are placed by clicking on their respective button. However, the button will not respond if the player does not have enough gold. This is achieved through a conditional if statement. The dragon can be placed anywhere on the map, but placing it on, or far away from, the enemies would be detrimental to the player, which discourages this. When a dragon egg is placed, it begins a timer. When the timer interval is achieved, the dragon will hatch, allowing it to begin firing.

4.2.5 Targeting

Dragons do not shoot at enemies when they are not hatched due to a Boolean condition which must be satisfied before targeting is allowed. Another method using collision detection prevents towers from being placed on one another.
If a dragon is placed too close to the enemies, it is seen as cheating, and it will not fire. Similarly, if the dragon is placed too far away from the enemies, they will be out of range. Therefore, the player will learn that the bookshelves are the best place to hatch the dragons.

4.2.6 Collision Detection

There are three kinds of collision which must be detected in the game. One of which is when a bullet collides with an enemy. Each bullet checks if it is colliding with an enemy by creating a QList of QGraphicsItems, and using a dynamiccast function to identify the enemies. If it collides with an enemy, it subtracts its damage from the enemy, and is then deleted.
Another collision type is when a player attempts to place a dragon over another one. The placement code then refuses to place the egg until the player picks an acceptable location. The collision detection works the same way as with the bullet.
The final sort of collision is when the shadow enemies reach the scrolls. Technically, this is not a collision, yet it resembles one. When the enemy reaches its destination, the crystal variable is decreased, and the enemy is deleted.

4.2.7 Spawning Enemies

Two timers control the spawning of enemies. Together, they act as nested for loops. This creates a certain number of enemies in a certain number of waves. To keep the game from becoming boring, more enemies spawn per wave, and after a certain number of waves, the

enemies become stronger. However, they drop the same number of crystals. This curbs the player's income.

4.3 Class Diagram

A class diagram describes how the objects and classes in the program are dependent on one another. The resulting hierarchy makes it easy to identify base and derived classes.

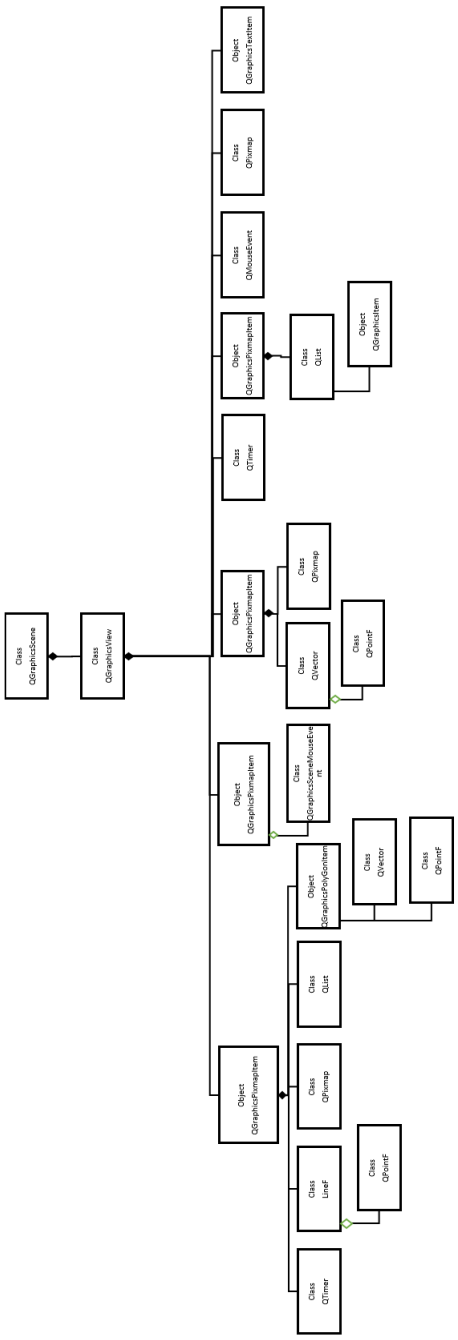

Figure 4.6: The basic class diagram for the game Scroll Keepers

5 Results

The result of this practical task, Scroll Keepers, is a simple tower defence game, which features all-original sprites. It successfully implements A* pathfinding, targeting, collision detection, dragon placement, resource acquirement, health deduction, and increasingly strong enemy spawns.

The game behaves largely as desired, with the exceptions of networking and selective tower placement, and animation in different directions. The bullets also become slower if there are many items on the map. A screenshot of the game in an early level is provided in Figure 5.1.

Figure 5.1: Scroll Keepers at an early stage in the game

6 Conclusions and Recommendations

The game's best features are probably its unique background and character sprites. They have a limited colour palette, and are all drawn roughly the same size. This, along with identical brightness and saturation levels, ensures that they give the entire game a consistent, harmonious graphics style. Furthermore, they are animated for additional charm.
Another aspect which the game performed well in was the clarity of the HUD. It clearly displayed the amount of available resources in high-contrast text. Also, the egg costs were displayed neatly by their respective eggs. Overall, the GUI is intuitive for most players.

The shadow enemies were spaced out evenly, which made the game flow fairly well. There are no enemies walking into each other, and they are all consistently animated. However, they do not change their sprites with their direction, even though the sprites facing the other way were included in the game. If this could be corrected, the game's visuals would be more appealing. Dragons target the shadow enemies with ease, and fire bullets with equal skill, but if there are too many items in the scene, the bullets slow down. This is due to the combination of the collision detection method used, and the fact that each background tile is also a QGraphicsItem. If the background could be simplified, the bullets should move quicker, improving the game flow.

A simple, subtle improvement to Scroll Keepers would be to include enemy polymorphism. The dragon classes implement this technique quite well, and the program would be much more readable if the enemies were separated as well. There were also a few other enemy ideas which could have been included, but were left out due to time constraints.
A networking component would greatly improve the entertainment value of Scroll Keepers. The best way to implement this would probably be to make a leaderboard where several players can monitor each others resources and remaining lives, along with their names. A simple sorting algorithm would keep the best score on the top of the leaderboard. The variables of the player's name, and their resources would be sent using TCP. This would make the game more competitive, and boost incentive to play.

Another feature which would improve the game would be a main menu where you can read Scroll Keeper's lore, check the rules, and start a new singleplayer or multiplayer game. Not only would this make the game far more user-friendly, but it will make Scroll Keepers look more polished.

Altogether, Scroll Keepers is a largely functional tower defence game, but it could have been greatly improved upon if the task had been started sooner. Not all of the objectives were met, but many were. Also, the objectives which were met were generally of a decent standard.