Documentation

Juho Karhu, Eero Saarro, Niko Somila, Patrick Sothmann

Overview

We created a tower defence game in which the objective is to place towers on the map so that the waves of balloons can not reach the end of the map. We implemented four different tower types with different damage values, prices and range values. Two tower types also have special features, such as the freezer tower which slows the balloons within its range. Towers can be bought and sold between rounds, and each tower can be upgraded once, enhancing its abilities.

We implemented 5 different balloon types and one ‘boss’ balloon. Each balloon has specific hitpoints, speed and armour values. Some of the balloons also have special abilities, such as the armoured balloon, which turns into a basic red balloon once its armour breaks. Popping a balloon awards the player with one money that can be used to purchase and upgrade towers. The game has two different map options which are read from a text file. and the balloon waves are also read from a text file.

Software structure

Overall Architecture:
The project has an entry point in ‘main.cpp’ within the ‘main()’ function. This initializes a ‘MainWindow’ object and displays it. The classes ‘Map’, ‘Projectile’, ‘MapTileGraphics’, ‘Defense’, ‘Balloon’ handle specific game functionalities. ‘MainWindow’ handles the Start menu GUI and its functionalities. The `Game_engine` class controls the game logic and interactions between various game elements. The main type of container used was the QVector. For example map tiles and balloon waves are stored in 2D QVectors and Balloon paths are stored in 1D QVectors.

Class Relationships:
- ‘Projectile’ inherits from ‘QGraphicsPixmapItem’ and ‘QObject’.
- ‘Map_tile’ inherits from ‘QGraphicsRectItem’.
- ‘Map’ contains a grid of ‘Map_tile’ objects.
- ‘MainWindow’ inherits from ‘QMainWindow’.

- ‘Defense’ is the base class for all the different types of towers.
- ‘Balloon’ is the base class for all the different balloons the player must defend against.
- All tower types and balloon types inherit from ‘QGraphicsPixmapItem’.
- ‘Game_engine’ inherits from ‘QGraphicsView’.

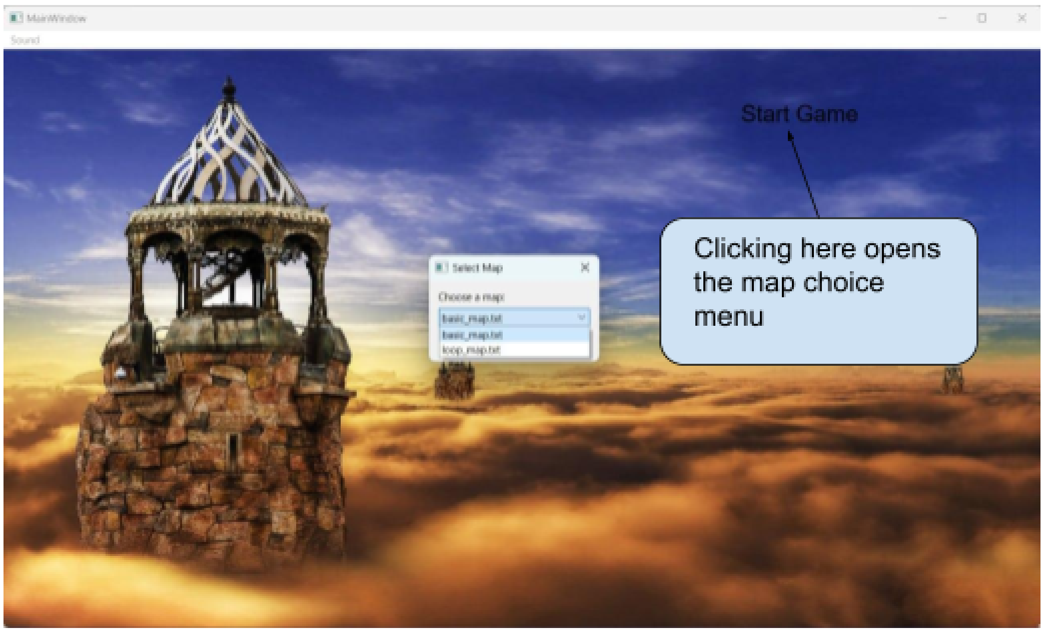
Interfaces to External Libraries:
We used the Qt framework for graphics, GUI, audio (‘QMediaPlayer’, ‘QAudioOutput’) and timers (‘Qtimer’). The ‘connect’ statement was used to handle events and communications between different objects. Tower attack speed and balloon movement speed was handled with QTimer.

Some Functionalities:
- ‘Game_engine’ controls tower placement, balloon spawning, waves, level progression, manages the game state (‘PlayerState’) using an enum, and click events.
- Tower ranges are of type ‘QGraphicsEllipseItem’.
- Balloons within range of towers are checked by using the collidingItems() method.
- Balloon targeting and attacking are handled in the specific towers subclass.
- Sound effects included are when balloons are destroyed and when towers shoot.

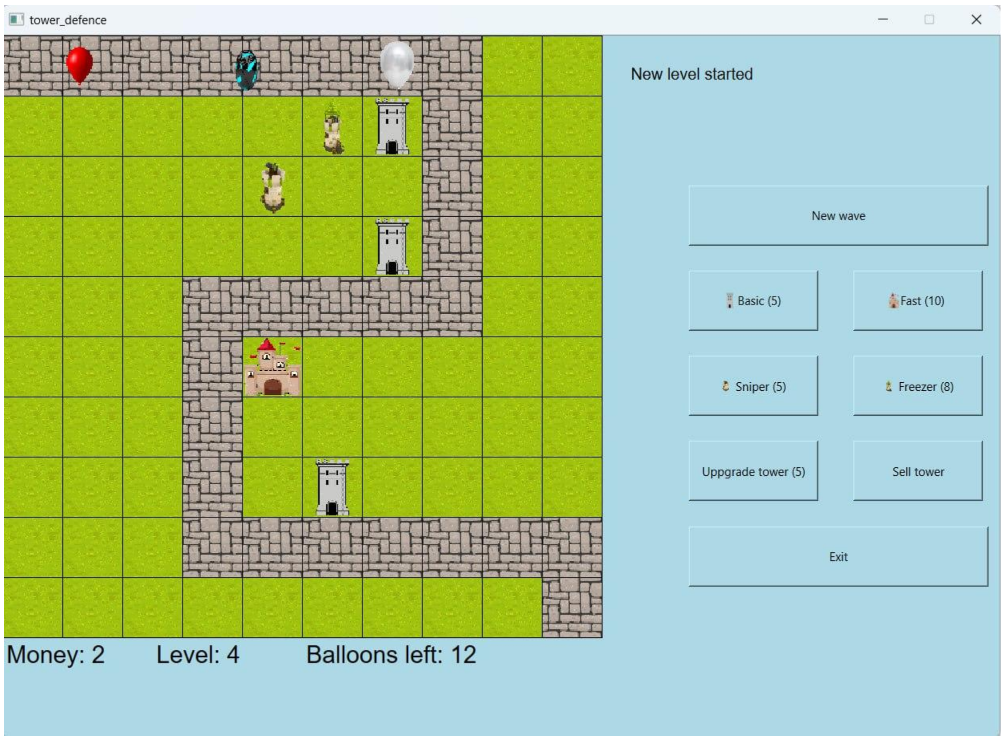
Instructions for building and using the software:
- Running the game requires the Qt framework (Qt creator with min Qt version of 6.0) and CMake (min version of 3.5).
- After extracting the project from the git directory, the CMakeLists.txt file should be opened in Qt creator and the project should be configured and built.
- Configuring and building can be handled within the Qt creator interface
- Qt should also include the ‘QMediaPlayer’ and ‘QAudioOutput’ packages as for some reason Qt does not always install these automatically with the Qt installation.

Playing the game:

Opening the game greets the player with a start menu where the player can select the map they want to play



After selecting the map, the game starts. The game window shows the amount of money, wave number and balloons left to spawn on the bottom part of the screen. Towers can be bought, sold, upgraded, the next wave can be started and the game can be closed with the buttons on the right side of the screen. The tower prices are shown on the tower buttons. Towers are bought by clicking on the button of the specific tower and then clicking on the specific map tile. After the tower has been placed, its range can be shown by clicking on it.



If a balloon reaches the end of the map, the player loses. The player wins by destroying all balloons before they reach the end of the map.

Notes:

- The sniper tower can only shoot balloons with armour on.
- The freezing tower slows down balloons within its range, except the silver balloon.
- The armour of the armoured balloons can only be destroyed with the sniper tower.
- The silver balloon’s speed is not affected by the freezing tower.
- Towers can be bought, sold and upgraded between waves.

Testing

We tested the game mainly by running it. We created different balloon wave files to check the functionalities of all the balloons and towers. We also printed out specific values to test the functionality of different scenarios. Some examples of testing were printing out values to test if a balloon is destroyed and then visually confirming the graphical results by running the game.

Work log

Week 1:

We went through the project plan with our advisor. Our next step was to begin installing the necessary libraries. The goal for the week was to set up the environment for everyone, enabling us to start coding. Our primary option was to install SFML; if that would prove challenging, we would switch to Qt.

Week 2:

Installing SFML on MacOS proved to be challenging and thus we decided to switch to Qt. We installed Qt creator and were able to run a simple mainwindow. The coding environment was set up for everyone and we could start implementing the code.

Eero and Juho started working with the basic classes and implementing a simple UI for those. Patrick and Niko started to implement the map and figure out the best way to create it.

Week 3:

This week was mainly used to experiment with Qt to figure out how we would use its different properties on our project. Eero and Juho added the base classes and were able to create push buttons with signals to start the game. A red balloon was added to the windows to indicate that pushing the start game button works.

Next week, Eero and Juho started to implement the starting view (graphics) with relevant buttons for playing the game. The goal for the next week was to have the relevant buttons and actions ready so we can start to work implementing levels and get the balloons to move. Patrick and Niko would implement the map and then start to think how to move the balloons in the map etc.

Week 4:

We were able to create the map and add balloons to it. We also implemented a timer for the balloons which moves them 1 pixel at a time along the path. We were also able to add towers to the map grid.

Afterwards we started to focus on how the towers can target the balloons. Meanwhile Eero continued to implement the spawning of different balloon waves to the grid after start level is pushed.

Week 5:

We now had first version of the game that was in a playable state. We also did some minor bug fixing and created a new map file with a looped path. Then we started designing and implementing the different tower and balloon classes. We started with the Basic tower and we were able to make it target balloons and shoot. We decided to use the collidingItems() method, because it seemed the most simple choice.

Week 6:

Our goal for the week was to have the game finalized for the demo

Juho added sound effects for the balloons and towers, Eero made some final graphical designs and Niko worked with finalizing the tower classes. The final tower that was completed was the freezer tower.

We added the money graphic and also implemented the increase of money by popping balloons and the decrease of money by spending it. We optimized wave difficulty and adjusted tower properties (fire rates, damage, etc.). Patrick started working on the demonstration slides.

Week 7:
The final week included the demonstration of the project. Eero, Niko, Juho completed some final fine tuning on the project code and completed final tests and bug fixes. Patrick worked on the documentation of the project.

We worked roughly 16 hours a week on the project. We set up a Zoom session link, which was always open, where we worked together on the project. We also created a Telegram group in which we updated each other on individual progress completed.