Tower Defense Group 16

Joonas Savola

Mikael Laine

Lotta Ketoja

Niklas Pigg

4.11.2021

General description

Our topic is the tower defense game. The main objective in this game is to defend against enemy attacks that happen in waves. Enemies move along a single predetermined non-branched path and their objective is to reach the end of it. By placing different defensive objectives such as towers, the player tries to stop the enemies from reaching for their goal at the end of their path. If enemies reach the end of their path, the game is over, and the player loses.

The game begins with an empty map that shows the attack path of the enemies. In the beginning, the player has some money to build its first defenses. By destroying enemies, the player will gain more money that can be used to build more towers and possibly upgrade existing towers.

Each tower has a fixed range that they can shoot. If the enemy is in the range of the tower, the tower will shoot the enemy. The towers will prioritize enemies that are closest to the end of the attack path. There will be multiple types of towers that will have different attacks and specialties. We will implement at least the required three different types of towers. Currently, we are planning to implement a basic tower, a cannon tower, a pulsing tower, and a freezing tower. The basic tower shoots one projectile at a time to one enemy. The cannon tower is like the basic tower but does more damage. The pulsing tower will shoot projectiles all around it with a fixed time interval. The freezing tower will shoot projectiles at enemies doing damage and slowing them. Each tower costs a different amount of money. The player can place towers on the map and change their position between enemy waves.

There will also be at least three different types of enemies. Currently, we are planning to implement a basic enemy, a strong enemy, and a fast enemy. The basic enemies have no specialties. As the waves progress the basic enemy will have more health. The strong enemy has a shield that must be destroyed with cannon towers. After the shield is destroyed it will turn into a basic enemy. The fast enemy moves faster and is harder to target. We will also possibly implement an enemy that splits into two weaker enemies when it is destroyed. The health of enemies will be implemented with different colors.

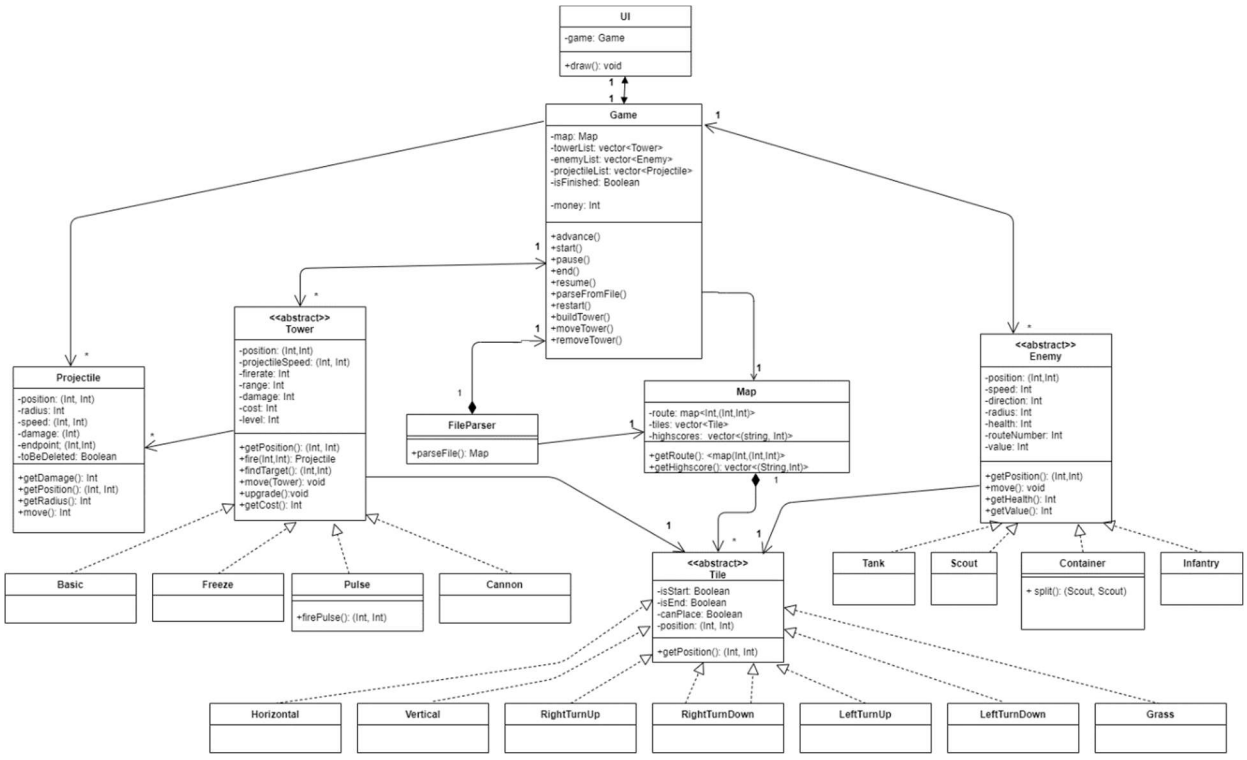
The game will include at least five different levels with different attack paths increasing in difficulty.

The game will be played with a mouse through a graphical user interface. The graphical user interface will show some information such as current score, money, and the number of enemy waves.

In addition to the required basic features described above, we will implement some of the additional features. Currently, we plan to implement at least upgradeable towers, more different kinds of enemies and towers, non-hardcoded maps, sound effects, and a high score system.

Class structure

The game functionality is comprised of main classes UI and Game, that handle the graphical user interface and game logic, respectively. In addition, the Game runs one Map-object at a time that simulates the entire game on a grid level. The map grid is comprised of objects of several subclasses of the Tile-abstract class. The map grid will be populated with Tower and Enemy objects as the game progresses. Both are abstract and their subclasses will be the actual towers and enemies on the map. When a tower shoots at an enemy, it creates a new Projectile-object targetting the enemy. Furthermore, there is a FileParser that parses pre-made documents into working game scenarios.



Picture 1: UML -class diagram

Algorithms and data structures

The enemies move along a pre-set path that is defined in the route-vector contained in the Map-class. The enemy's speed determines how many steps along the route it moves at a time.

The targeting iterates over the enemies in the container of the Game-class, finds the one that is the furthest along the path, and if it is in range locks on to that enemy, it then finds a point in the enemy's route, where the time the enemy takes to get there and when the bullet gets there are equal, and outputs the coordinates to that point.

When a target has been found, the tower fires at the coordinate outputted by the targeting, creating a new Projectile-object headed in that direction, the Projectile travels until it hits the enemy, or until it runs out of the tower's range in the case of the Pulse-tower, thereby marking th its toBeDeleted as true, and at the next advancing the game deletes this Projectile.

When a container enemy is hit and its health reaches zero, it explodes and creates several scout enemies in its place.

Planned use of external libraries

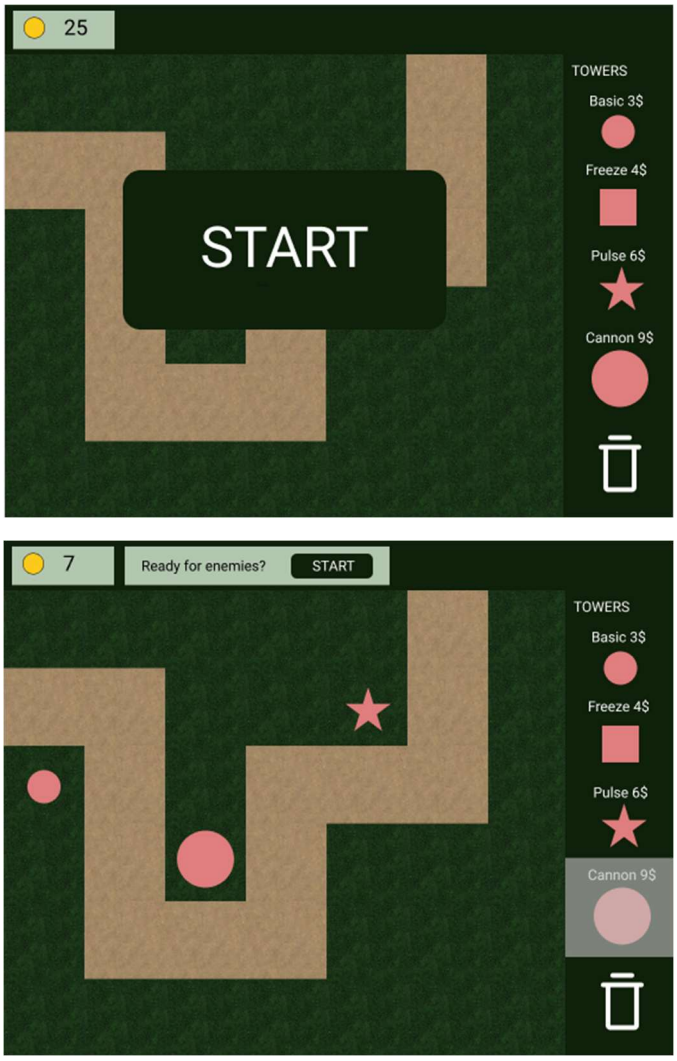
For the graphics, we plan to use the SFML library [1]. We will possibly use some collection algorithms from Boost [2]. We will also possibly use Box2D [3] to implement some physics elements, such as object collision, but we have not decided yet. We will take a closer look at the documentation of those libraries before we decide whether we use them or not.

Testing plan

After implementing new parts of the program, we will test the implemented features and classes by doing a simple executable for each class. In these executables, we will make sure that the functionality works as intended and that there are no errors. We try to do the testing as soon as the implementation is ready to catch the errors early. This also helps to locate the errors better when the tested part is relatively small at a time.

For each executable testing some class we will also run the Valgrind the check that there are no memory leaks etc. After the tests are successfully completed and there are no known errors left, the feature will be merged to the master branch.

Draft user interface



Picture 2: UI examples

Use case description

The user starts the game and the first thing shown is the start menu. The start menu will include buttons such as start game, high scores, and setup. The user clicks the start game button and a list of playable maps will be shown. After choosing the map, the user press the start button and the game starts. In the beginning, the map is empty containing only the attack path.

The user can place towers to the gameboard by “drag-and-drop"-method. The gameboard is built with tiles and a tower is always placed to the center of a tile. Towers cannot be placed on the path or top of each other. The cost of the tower is visible to the user and when the user adds a new tower, the money counter will be decreased accordingly. If the user doesn’t have enough money to build a tower, the tower won’t be draggable (grayed out). The user can also move a tower by dragging and dropping it to a new place. If the user wants to remove a tower, he can drag it to the trash can. The cost of the tower will be returned to the user. When user thinks he is ready for the enemy wave, he can press the start button on

the top of the window. While in “enemy wave”-mode, all the tower movements will be disabled. The money counter will be increased each time an enemy is killed.

After the game ends, the score will be shown to the user. There will be a replay button that starts the same map again and a button that takes back to the list of maps.

Planned schedule and division of work

Week 1: Planning and project setup

This week we all do project plan documentation and setup the project.

Week 2: Basic classes and their simple methods

Classes and their basic methods are implemented according to the division of work and bug tested accordingly.

Joonas Savola: Tower-class

Mikael Laine: Enemy-class

Niklas Pigg: Tile & Map –classes

Lotta Ketoja: Projectile-class

Week 3: Game-class skeleton, UI

This week we will focus on implementing simple versions of the Game-class and graphical user interface. We will also link the program logic and graphical user interface to work together.

2 people working on the Game-class

2 people working on the UI

Week 4: Advanced methods of Tower, Projectile and Enemy, finishing Game-class, and FileParser

This week we will implement more advanced algorithms, such as enemy and projectile tracking. We will also implement the FileParser-class that allows map input from files. The Game-class will be developed further.

2 people finishing the methods of the Game-class

1 person implementing the FileParser

1 person implementing advanced methods

Week 5: Finishing and bugtesting

This week we will do necessary optimizations, tweaks and bugtesting the system as a whole

Week 6: Documentation

Everyone documents their own classes and jointly developed classes are documented as best seen fit.

References and links

References used in this document:

[1] SFML. https://www.sfml-dev.org/

[2] Boost. https://www.boost.org/

[3] Box2D. https://box2d.org/

References that we plan to use when developing this program:

https://en.wikipedia.org/wiki/Tower_defense

https://en.cppreference.com/w/

https://www.cplusplus.com/reference/

http://opengameart.org