Tino Korpelainen 896421

Nuutti Nykänen 900650

Patrik Ahola 591111

Elias Kauranen 793595

C++ Tower Defense Project Plan

We are creating a tower defense game as a project for the Object-oriented programming with C++ course. In a tower defense game, a player is tasked with managing a set of purchasable towers that are placed on a map. These towers, different in nature, attack enemies that cross a predetermined path across the game map. The player is required to defend the game map exit from enemies – this means that enemies shall not reach the end of their path. The player wins when all waves of enemies are defeated. The player loses if enough enemies pass.

We will naturally implement at least the "base version" of a tower defense game. This means implementing a GUI that allows a player to interact with a store and place towers on the map. The map has an enemy path read from a file. Waves of enemies walk through the path as the player's towers automatically and independently attack them. The game contains different towers – for example, one tower can simply attack enemies, but another can enhance the abilities of surrounding towers. Enemies also differ – each has their own amount of health and one might spawn another enemy upon defeat.

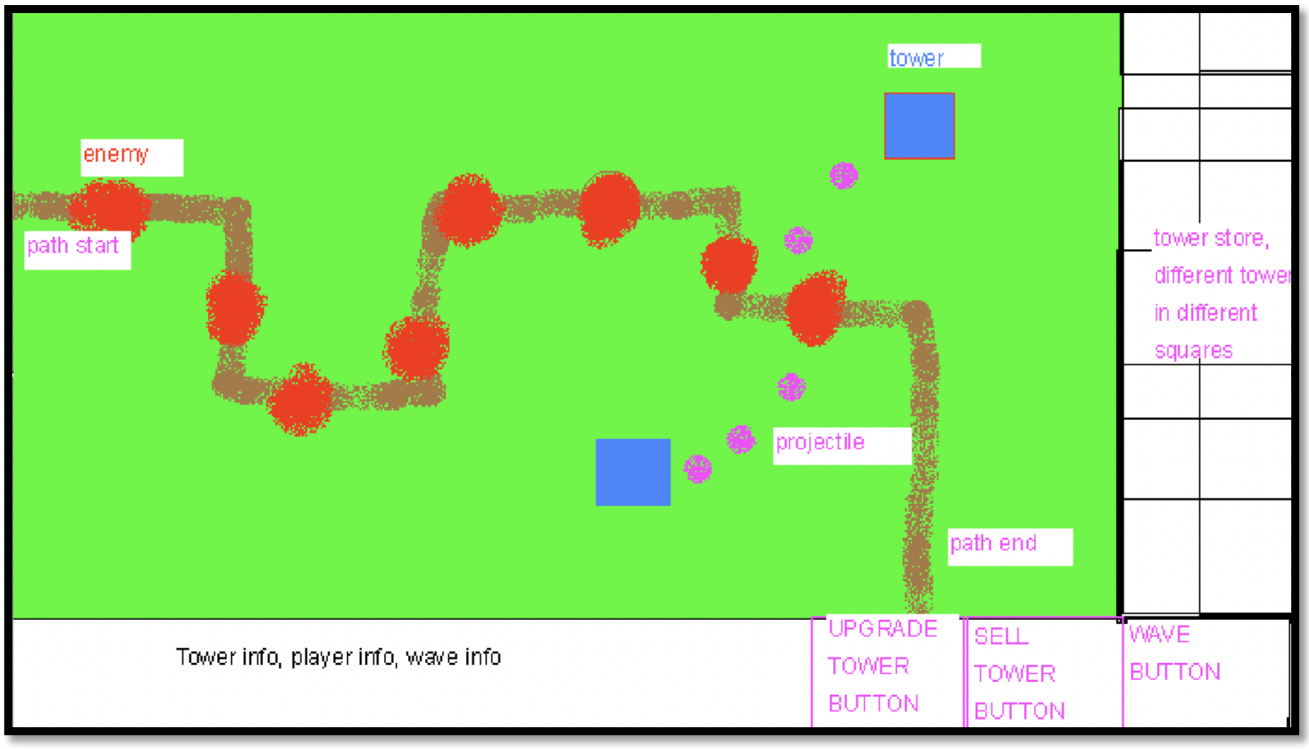
Additional features we decided upon include non-hardcoded maps. This means we will read different levels from a file. This feature allows for a level editor within a text file. The player can thus create their own levels by accessing the text file within the game folder. Another feature we will implement considering text files is a high score tracker. This allows for the player to see their past points written in a text file.

We will also consider more tower and enemy types than what is required. This should not be a hard addition and it makes the game much more fun and versatile. We will especially focus on implementing different kinds of towers as we see much potential in them. Towers will also be upgradeable in our game. This means investing more money in a tower to make it stronger. We will also implement a strength / weakness system between towers. This means that some towers will be stronger than usual against certain enemies.

We will make it possible to place towers mid-wave, which allows for quick responses from the player and requires constant attention. The last additional feature we will consider is the implementation of sound effects to make the game livelier.

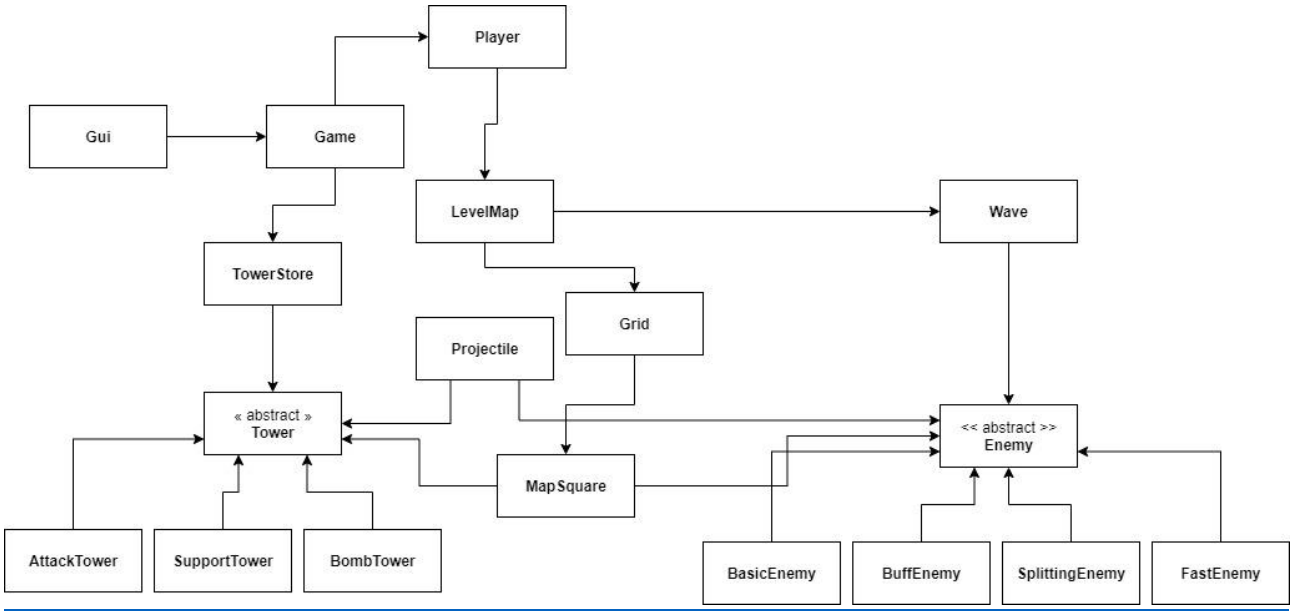
The game starts with a map that contains a predetermined enemy path. For the player to start the actual gameplay, they are required to make a call for the game to launch a wave. This kickstarts an unstoppable wave of enemies that ends only when the wave's enemies either are defeated or they get to the end of the enemy path. As stated before in the additional features, the player will be able to interact with the store and the map mid-wave (also between waves).

All interactivity is done via the GUI, via clickable buttons. The buttons' functions are clearly stated by text inside them, for example "START WAVE". The store is also image-based, which means the towers are displayed in a grid at the side of the GUI. The game ends when either the player wins or loses. This prompts the game to asks a username that it uses to record the player's score to a text file for later inspection.



Game interface

The high-level structure of the software



GUI

- Shows (draws) the game along with the map and everything in it. Listens to the game and updates accordingly.

Game

- The game itself, shown by the GUI. Contains all the elements in the game (enemies, towers, map etc.) Responsible for spawning such elements, starting and ending waves or the game, progressing time I.e., running the game.

Wave

- Contains a vector containing enemies. These enemies spawn at the start or during the wave.
- Read from a text file.

Player

- Contains the player's money and health.
- Contains a list of the player's current towers.

Grid

- A two-dimensional table of all the MapSquares in the map.

MapSquare

- A map square with x- and y-coordinates. Can be an enemy path, contain a tower, be empty or be unbuildable.

LevelMap

- Contains the Grid, towers in the grid and enemies in the grid. Also contains a linked list of the enemy path.

Tower

- Abstract base class for three types of towers: AttackTower, SupportTower and BombTower.
- Towers have a name, cost, range, location on the map, sprite, description.
- Each tower type has a specific projectile along some features specific to the type of the tower.

Projectile

- Shot by a tower, has a speed, damage, sprite (a picture representing the projectile's appearance) and an effect on collision.
- Knows what tower launched it.

Enemy

- Abstract base class for different enemies, BasicEnemy, BuffEnemy, SplittingEnemy and FastEnemy.
- Enemies have health, speed, moneydrop value, sprite, status, location on the map, immunities specific to the enemy type.

TowerStore

- Has a vector containing towers, the player can buy towers from this vector, having the sufficient money.

The planned use of external libraries

We are planning on using SFML and JsonCpp. SFML provides us with a graphics interface, listens in on player input and allows us to play sound effects / music. JsonCpp allows us to interact with

.json -files which are needed for saving and editing levels and high scores. We shouldn't need any other libraries since SFML provides so many functionalities.

Division of work and responsibilities between the group

We agree on weekly responsibilities in our weekly meetings.

Our division for week 1:

Tino:

Creates a prototype "hello world!" -program which has working imports and Makefile. Prototype creates a window and does rudimentary .json -editing.

Patrik:

Creates a first version of the class Game, the class will be constantly updated throughout the project as other classes and further functionality are implemented.

Elias along with Nuutti:

Laying out the basis for a game map.

Planned schedule and milestones before the final deadline of the project

5 weeks all together

1 week:

- A working prototype SFML -program which everyone can open.
- Working library imports and a working Makefile
- First version of class Game. It will be updated all the time during the project.

2 week:

- Map

- Working enemies and correct movement

3 week:

- We have a basic map and enemies moving (week 2)
- Now we add the towers and player stuff
- GUI

4 week:

- Market/Shop, player stuff
- Reading from file and writing to file?
- Sound effects