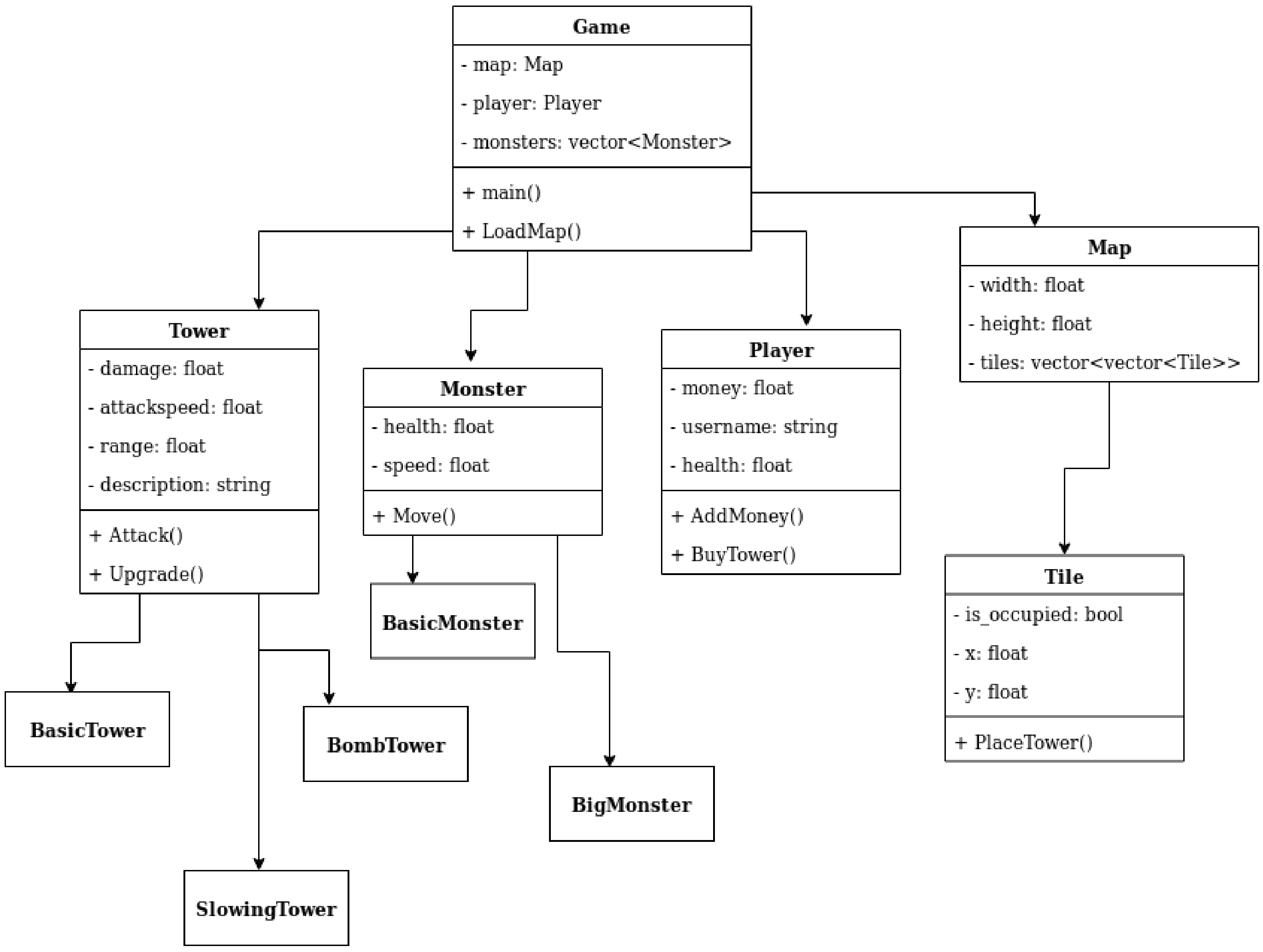
Project plan 28.10.2019

Project description

Tower Defence (or informally TD) is a subgenre of strategy video game where the goal is to defend a player's territories or possessions by obstructing enemy attackers, usually achieved by placing defensive structures (towers) on or along their path of attack.

Architechture

Here is a rather simplistic UML diagram explaining the different classes and how they relate.



The main class is Game. It keeps track of the game state and contains the main functions as well as the graphical user interface.

We have abstract classes Tower and Monster which have subclasses that represent different kinds of towers and enemies. The Map class consists of tiles that keep track of towers and monsters. Class Player includes the player information i.e. how much money he/she has, username, health etc.

Schedule

We will try to implement the basic structure for the game as soon as possible, which will allow us work more independently on various features. Preliminary after this, our workflow would mostly consist of opening git branches and making pull requests or just simply making git commits to the project. We could do occasionally meetings whenever we feel like we're in need of one.(We see each other anyways several times a week, so probably few arraned meetings are needed).

We will also try use issues as a TODO list, to keep track of features needing to be implemented and assigning members to certain issues. Issues will also allow us to have discussions regarding implementation of the feature instead of having to schedule a meeting.

Before the mid-term meeting we aim to have a working implementation of our Tower Defence game with basic features and a simple GUI. Eventually we could strive for better looking graphics and more advanced features.

Group roles

We don't have any specific group roles as of yet. Markus will be the project manager and plan meetings and keep track of our TODO list and schedule. As for now we will pick issues we would like to work on and implement the necessary features.

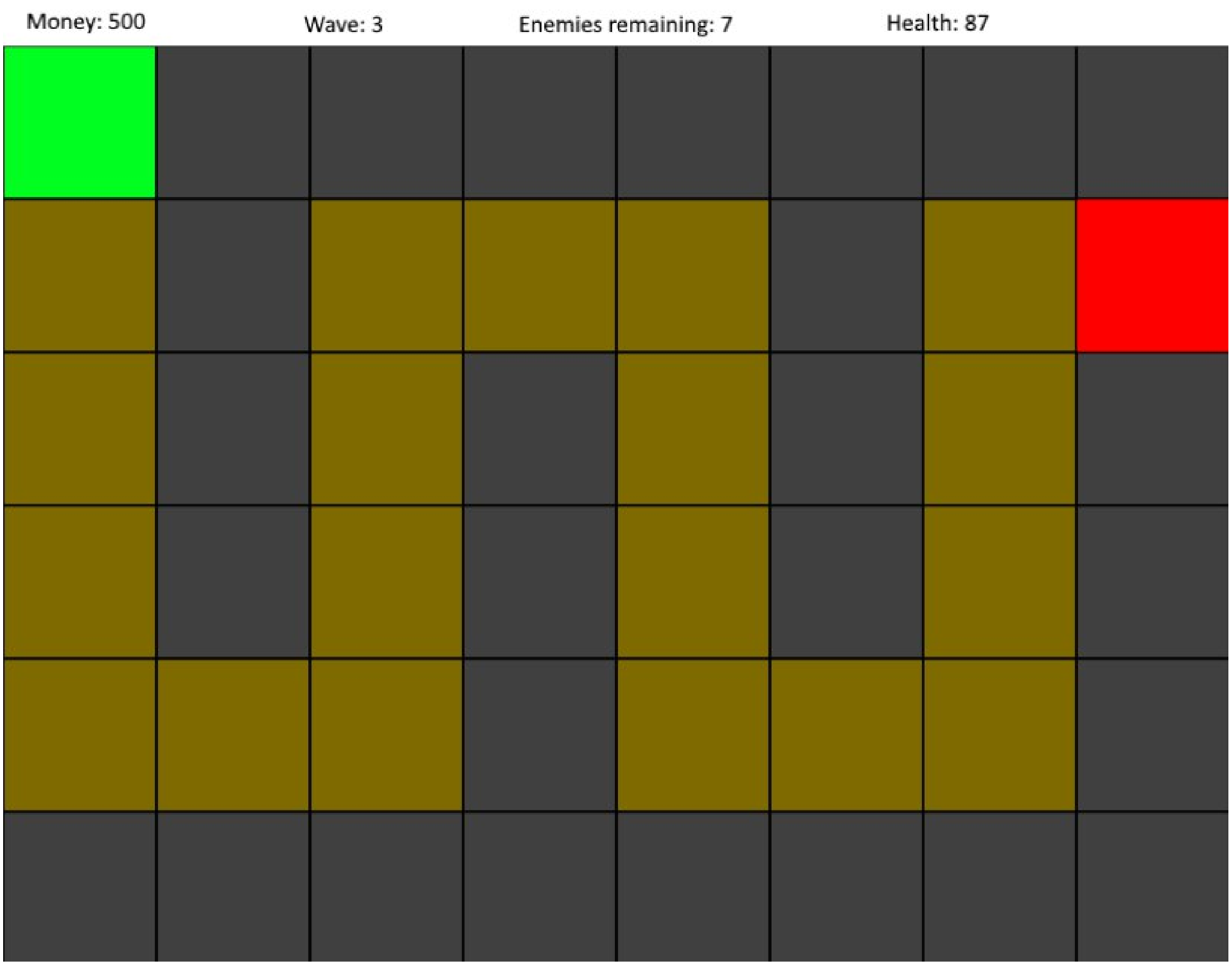
Features

In addition to all the basic features, we are planning to implement the following advanced features:

- **Non hardcoded maps (read from file)**
  - The maps will be read from a file that will be in text format. The file will consist of a grid of symbols. Each symbol will define what each tile contains. The user should be able to resize the map to almost any size just by adding another row or column to the grid in the file.
- **Upgradeable towers**
  - The towers should be upgradeable for a specific cost depending on the tower and current level of the tower. Upgrading a tower may increase the damage, range and attack speed of a tower and could possibly add some special features to the tower
- **More kinds of enemies and towers**
  - We will make a few different types of enemies and towers. The different kinds of enemies will have different health or speed. The various kinds of towers will have different damage, range and attack speed.
- **A list of high scores that is saved locally per map, with a username**
  - The high scores will be saved to a file which contains each map with highscores for each user that has played that map. The user will be able to view the highscore list ingame.
- **Dynamic enemy paths that are altered with the placement of towers**
  - Dynamic enemy paths will be implemented using the A* search algorithm, a more thorough explanation is in the Algorithms section of this document. The players should not be able to block the path so that the enemies can't reach the target.
- **Sound effects**
  - We will implement different kinds of sound effects for various things in the game. For example enemy attacks will have sounds and buying or selling a tower will also have a sound.

Graphical user interface

A simple user interface keeping track of lives, wave number and ingame currency. Towers are bought from a menu on the side. Below is an example of how it might look like:



The game window will consist of the map drawn in the middle, with information about the current game situation shown to the user at the top of the window. The user will be able to see the various towers available for purchase with relevant information for each of them, such as price and stats. By clicking on a tower on the map the user will be able to see information about the tower and either upgrade or sell the tower.

Algorithms

One algorithm that will be needed is the A* search algorithm. It will be used for the dynamic enemy paths. Since we're using a grid system for our tower defence map it is a suitable algorithm to use since it's easy to implement and works efficiently. The algorithm will have nodes assigned to each map tile and will calculate the cost of the path to each tile and find the path with the lowest cost to the target tile. https://en.wikipedia.org/wiki/A*_search_algorithm

We will also make use of Pythagoras' Theorem to calculate the distance to the closest enemy. The towers will attack the enemy closest to the tower. So to accomplish that we simply loop over each enemy and calculate

the distance and choose the closest one. It may not be most efficient way of doing it but in our case we have such a low amount of towers and enemies that it is a suitable solution.
https://en.wikipedia.org/wiki/Pythagorean_theorem

Libraries

We will use SFML for the rendering and sound effects. We chose SFML instead of SDL since it seemed like a more higher level library. Thus we can spend more time implementing useful features instead of trying to implement higher level functions for using SDL. SMFL also has pretty clear API documentation so we should be able to find documentation for all the SFML features we need. SFML seems to have all the required features for the tower defence game

For building the project we will be using CMake since it was recommended and it should let us get cross-platform compatibility in our project relatively easily. CMake will also make it easier for us to manage libraries and dependencies, while keeping the project platform-independent.

We might use Boost for more platform agnostic functionality that STL is missing. One use we would have from Boost is parsing setting files. Boost includes a Property tree which would allow us to parse XML, JSON, INI files, file paths in a rather simple way.