ELEC-A7151
Object oriented programming with C++

Game: Tower-Defense Group 3 Project Plan

Daniel Nikkari, 653088

Sani Letchu, 715036

Oskari Kiili, 728890

Vili Nieminen, 593465

Game: Tower-Defense

Daniel Nikkari

Sani Letchu

Oskari Kiili

Vili Nieminen

Table of contents

Table of contents **2**

1. Introduction **3**

2. Scope of the work **3**

3. Program structure **5**

4. External libraries **6**

5. Development tools **6**

6. Division of work and responsibilities **6**

7. Schedule and milestones **8**

8. Final words **8**

References **9**

1. Introduction

This project will implement a tower-defense game using C++ programming language. “Tower defense (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 on or along their path of attack.” from Wikipedia article *Tower defense* (2021). In TD, enemies move in waves and try to move from point A to point B through paths indicated on the map, and the towers placed by the player try to stop them. The objective of the player is the survival of the base.

Players can add towers to the map by clicking the wanted tower and dragging it to an available location on the map. The towers will automatically shoot enemy NPCs who come to their range of fire. Players will have money that they can spend on buying upgrades for existing towers or buying new towers. The player loses when his base gets destroyed by the enemy NPCs, i.e. when they reach the end of the path.

The game includes multiple maps with increasing difficulty, and hiscores are kept on each map. The aim is to have the game running with more and more levels until the base is destroyed by scaling the number of enemies and potentially their properties.

2. Scope of the work

We took features from the topic description and split them semantically into three categories per importance: basic (required), additional and a wishlist if time permits. As a further heuristic, the bolded features are which we aim to implement and unbolded ones of secondary importance under the three main categories. Some of the additional features have points in brackets, copied from the project description, and the ones without are something we came up with ourselves.

Basic features

- **A functioning tower defense game with basic graphics:**
    1. **Enemies follow a single, non-branched path**
    2. **Towers can shoot enemies inside their range**
    3. **Game is lost if any enemy reaches the end of the path**
    4. **Some money system, which gives more money per enemy killed and money is required to build towers**
    5. **Two modes: placing towers, running a wave of enemies through the path (towers cannot be moved when enemies are on the map)**
- **At least three different types of towers, for example:**
    1. **A basic tower, shoots enemies within its range**
    2. **A slowing tower, slows down enemies inside its range**
    3. **A bomb tower, shoot a bomb when enemies in range, can kill multiple enemies**

- **At least three different types of enemies, for example:**

    1. **An enemy that's killed immediately when it's hit**
    2. **An enemy that takes multiple hits to kill**
    3. **An enemy that splits into multiple kind-1 after killed**
- **At least five different levels with increasing difficulty**
- **Controlling the game by mouse: user can build/remove towers either between waves of enemies or without restrictions.**
- **Simple user interface that shows information such as resources, number of waves/enemies etc.**

Shortlisted Additional features

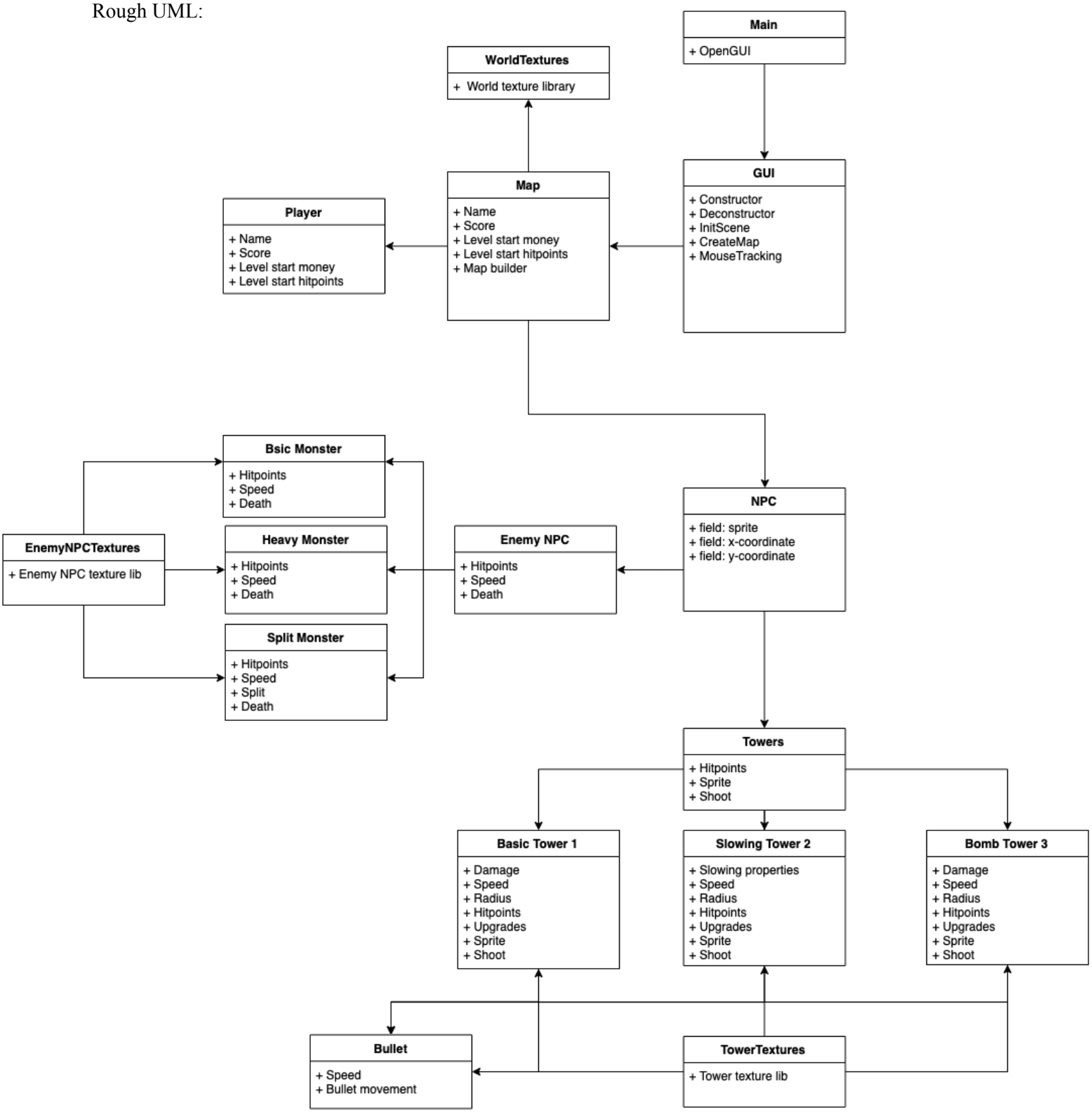
- **Non-hardcoded maps, i.e. read from a file (or randomly generated) (2 points)**
- **Upgradeable towers (2 point)**
- **More different kinds of enemies and towers (2 points)**
- **A list of high scores that is saved locally per map, with a username (decide yourself how to calculate points) (1 point)**
- **Sound effects (2 points)**
- Multiple paths of the enemies, so branched paths and enemies choose one with some intelligence (3 points)
- Dynamic enemy paths that are altered with the placement of towers (2 points)

Potential additional features if time permits

- Tower placement can be altered during an enemy wave (2 points)
- Level editor to create levels and to save them in a file (4 points)
- Towers can be damaged by enemies (2 points)
- Different attack and defense types for both the towers and the enemies. For example, some attack types may be more effective against some defense types. (2 points)
- Multiplayer: game can be played from a "client" and a "server side" stores the scores to one place (3 points)

3. Program structure

Rough UML:



4. External libraries

The project will most likely use SFML for game development. Couple of team members have previous experience with the Python extension of the Qt, but other than that, our team has no previous experience from the libraries and therefore have to do more research to make our final decision.

The chosen testing framework is Catch2 due to its simplicity. Most of the classes and their content should be covered by unit tests, and the GUI-specific elements are planned to be tested by conventional manual testing.

5. Development tools

The project uses different tools to create a robust development workflow and ensure that the ready software meets its requirements.

The backbone of this process is Git with Gitlab for version control. Our group uses a simple branching process: a develop branch parallel to master, from which everyone branches their feature development. After suitable intervals (like the weekly meetings) the progress in the develop branch is reviewed comprehensively and pushed to master. No one in the team should push to master themselves unless by accident which is then most likely to be reversed and pushed to develop instead.

Doxygen is used to produce automatic documentation of code. The group is currently unsure whether it is feasible to also readily produce documentation of test results, but this would be a handy addition instead of copying results manually to a Markdown file. In any case, these diagnostics are most likely very short.

Finally, Make is used as the build tool of the project. This choice is based on our team's limited experience with makefiles and no experience with CMake. CMake can be considered at the end of the project for portability between different systems if time permits, but at this stage its incorporation seems unlikely.

6. Division of work and responsibilities

We will begin by making the first versions for main.cpp and gui.hpp and gui.cpp, furthermore, we will make the first version of map.hpp and map.cpp somewhat together. We aim to do this like described because we want all of the team members to understand the most important part of the game mechanics properly. Moreover, we will later divide the making of map version two between the four of us with a tentative split:

- WorldTextures (Oskari & Daniel)
- Maps from file (Sani & Daniel)
- Player, mouse tracking and GUI (Vili)

For NPC methods and functions,

- NPC, bullet classes and basic functionality (Vili & Oskari)
- Sprite functions, when sprite sheet textures are ready (Daniel & Sani)

- Textures (from the internet)

At this point, we should have a majority of the functionalities related to the game. Then we can assign more specific responsibilities related to arising issues when needed.

The work and responsibilities here are a tentative sketch and of course will live with the project as required.

7. Schedule and milestones

wk44 - 5.11.2021	Project plan return deadline, find textures, check that everyone can compile a program with SFML.
wk44 & wk45 - 1.11.2021-14.11.2021	Get acquainted with Doxygen, SFML and Git. Install cpplint, setup Git and git branching. Begin first coding
wk46 - 15.11.2021-21.11.2021	1. version main, gui & map and better work division.
wk47 - 22.11.2021-28.11.2021	2. version of main, gui & map; the foundation should be solid at this point. Start NPC and sprites
wk48 - 29.11.2021-5.12.2021	Finish game elements. The game should be running to some extent at the end of this week
wk49 - 6.12.2021-12.12.2021	Polishing, additional features. **Project return deadline**

8. Final words

Questions:

- Is it mandatory to be able to move towers?
- Can we use QTdesigner?
- Is detecting enemies with Pythagorean's theorem in a tower's range the best method? Looping the distance between every enemy every tick. Seems naive but could work well as long as there are not too many enemies (O(nm)).

References

“Tower defense”. (2021). Wikipedia. Available at: https://en.wikipedia.org/wiki/Tower_defense