Project Documentation

Tower Defence Tran Duong 6

Aleksanteri Pitkenin 674944
Antti Tiusanen 742261
Håvard Kalliainen 10 1192192
Nia Lehtonen 939773

Overview

The finished project is a tower defense game with a cat and mouse theme, where the towers represent cats and the enemies are mice. Running the main function launches the GUI, which starts from a main screen. The main screen presents two options: 'start' and 'exit.' Choosing 'exit' will naturally close the GUI, while selecting 'start' begins the game.

The game layout features a sidebar on the right that displays all three available towers for purchase, along with the player's money. Therefore, the wallet system is fully implemented there. Players can purchase and place a tower at any time during the game. Two towers cannot be placed on top of each other, and players cannot purchase a tower for which they do not have sufficient funds.

The game map consists of tiles, and there is a specific path for the enemies to traverse. Towers can only be placed on empty tiles, which are those not part of the enemy path, rocks, or tiles that already have a tower.

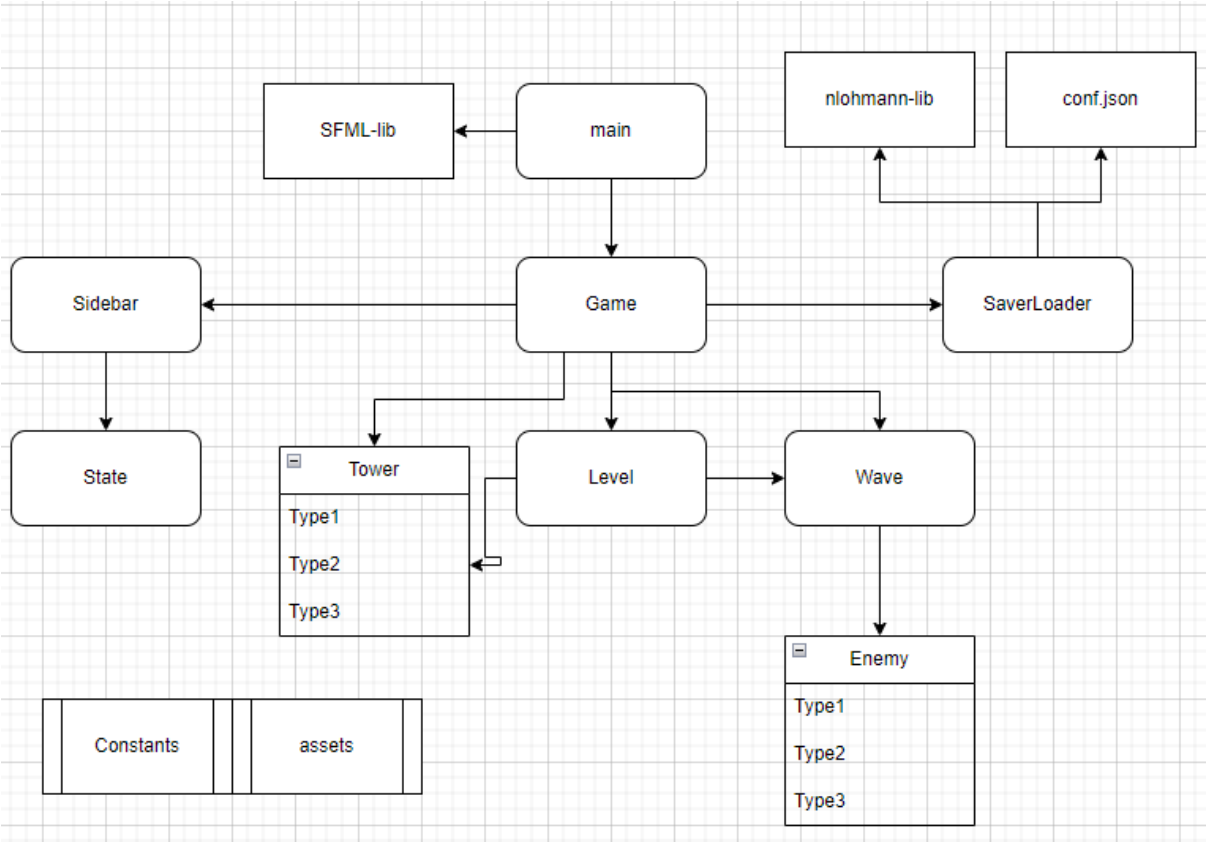
There are three types of towers in the game. Progressing from level 1 to 3, the first tower slows down enemies, the second one scratches enemies on neighboring tiles, and the third one can shoot enemies from a longer distance. The level 1 tower is the least expensive, while the level 3 tower is the most costly.

Additionally, there are three types of enemies. The first is the basic enemy, characterized by standard speed and basic health points (HP), and it is the smallest among all the enemies. The second type is a fast enemy with fewer HP than the

basic enemy, while the third type is a slow enemy with the highest HP. Enemies spawn on the map in waves, with the first level having the fewest enemies and waves, and the fifth level having the most.

The player wins a level by destroying all the enemies. Five levels need to be won in order to win the game. The player loses if even one enemy reaches the end of a path. Once a level is lost the player has the option to either exit the game or restart the level.

Software structure



The architecture provided in this UML-shows class relationships pointed by arrows. For example main used an instance of Game which uses instances of Sidebar etc. main also uses the interface to SFML-library and SaverLoader which is responsible for the configuration and I/O-stream used nlohmann-library and the configuration file. Tower and Enemy classes have different variations or types. Constants contain variables that are used

in every other class so that they can be easily changed among the file system. Assets contain pictures and graphics.

Instructions for building and using the software

After cloning the repository the software requires you to run the command **“./setup.sh”** Which installs the required dependencies. Then “**make**” to run CMake which builds the program. Lastly “**./main**” is used to run the program. Start button begins the game. The money is shown in the right bottom corner and by pressing the towers and placing them on grass tiles using the UI you spend the amount of money that is the tower cost and the towers start shooting at the enemies. You win the game by defending against the enemies through the waves and lose if a single enemy crosses through the path.

Testing

All of the testing of the program was done by manual testing since we were able to build and run the program at around the same stage when implementing unit testing was relevant. Console shows print commands during the run time which is used to determine some value based tests. Otherwise the functionality working properly during runtime has been deemed to be sufficient to prove that some aspects of the game are working as intended. Outcomes for all modules of the programs were positive in the end. During implementation the manual testing (for example the GUI not working properly) was pretty evident in showing defects in the program run and then console prints were used to locate the source of these defects and to check module and object values during the runtime. Detailed descriptions on how all of the modules were manually tested are in the tests/ folder. After making merges to master the changed file tests were revisited to see that everything works as before. Constants and assets do not have separate manual testing as it was included in everything else.

Work log

Everyone was busy during the period so we divided the original work and changed it up a little during the first weeks. After making our initial work we started working together more during week 49. Before that the responsibilities were divided into the following. At this point everyone had used about 10-15 hours on the project.

- Antti: I/O stream, class structures, level editor, details on the architecture
- Aleksanteri: connecting what everyone else was doing into a runnable prototype
- Håvard: User interface research and implementation
- Nia: Main logic, initialization, running main, creating instances, nlohmann

We had a meeting on week 49 to determine how we finished the project. At this point we had a working prototype which was used for testing different branches. We also divided the remaining work to be done into bullet points that people could choose from, create a branch on and complete. During this last week everyone used about 20-25 hours for the project. The final work load went according to the following.

- Antti: abstract classes, documentation, testing documentation
- Aleksanteri: different tower types with separate functionality
- Håvard: updating documentation, reading levels from file, level progression
- Nia: documentation, testing documentation, abstract classes, cleaning code

And that is how we ended up in the final commit.