Project Plan

Tower Defence Tran Duong 6

Aleksanteri Pitkenin 674944

Antti Tiusanen 742261

Håvard Kalliainen 101192192

Nia Lehtonen 939773

Scope of the work

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

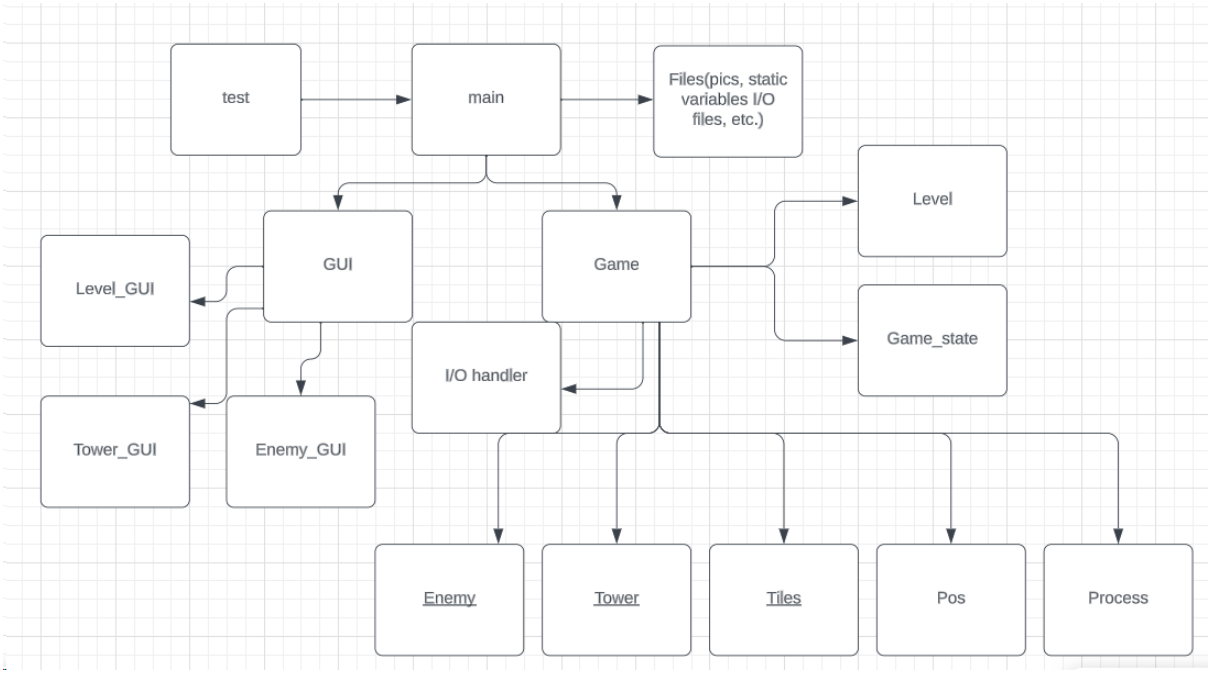
- Controlling the game by mouse: the user can build/remove towers without restrictions.

- Simple user interface that shows information such as resources, number of waves/enemies etc.

We will discuss adding additional features once the basic features are implemented.

High-level structure of the software

On the top level we will have the main program and the unit tests. Within the main branch we are going to have a directory for all additional files that are referenced in the main software, GUI and UI. Within the directory we will put pictures, txt files, and dynamic I/O files. The main branch will be separated into the GUI branch and a directory for all of the logical files (basically everything that is not the GUI). Enemies, towers and tiles on the map are going to have abstract classes which will be derived into the specific classes (different enemies etc.). The level is constructed of a matrix containing "references" to different tiles. A Game file will contain the main logic of the running game such as moving enemies, towers shooting, buying and other basic components that are included in TD game. One file will contain the current state of the game. A class for position will be included. This is used for the map and enemy/tower positions. I/O stream for saving the game state and loading it from a file will be included in a file. A wave class has information about the enemies that are included in a wave. It can have many instances for different waves. The GUI has a main GUI for all of the general stuff and sub-GUIs for enemies, the level and towers.



UML of the file hierarchy and file dependance. Underlined files are abstract classes. The arrows point files that are straight dependent on the previous files. We are trying to achieve a stack architecture to make group development easier and error detection easier.

Planned use of external libraries

- SFML: For 2D graphics and simple animations.
- Qt: If you plan to have more complex UI elements
- Standard Library: For saving game state, reading levels, etc.

Division of work and responsibility between members

To Do:

Enemy class, Tower class, Tile class, Money/shop/wallet??, Level design, Main logic

Aleksanteri Pitkenin - Level class, maybe also tower and enemy classes, goal is to have a basic (boring) working prototype

Antti Tiusanen - Additional details, class structures, I/O stream, level editor.

Håvard Kalliainen - UI, research Qt.

Nia Lehtonen - Main logic: Initialization, loading everything to main (create instances of everything etc. enemies, towers…)

Planned schedule and milestones before the final deadline

Week 0

During the first week we are trying to get answers to all of the questions that we are wondering about the project from Tran Duong. We will also try to create a “skeleton” or initialize the class structure/hierarchy with initially empty files with some comments on how they should work so people can work on different areas even if the critical files are not implemented for them yet. Investigate how potential UI can be created and familiarize oneself with viable libraries.

Week 1

We will try to get an almost complete basic structure so we can communicate with each other about critical class associations so we can start finishing our initial project structure towards a working prototype of the program. Make a UI sketch so that group members are oriented on how the game will look.

Week 2

Working game but not finished. We can create unit tests and the game should have at least a working wave, towers and shop. From this point we can start discussing the additional details and polishing the previous work. We will create a vision of the finished project.

Week 3-...

We will try to get finished/tested additional features for the project at this stage. Polishing previous features, discussion about cross-programming. This will be iterative and we will hope to have stable progress during the final weeks with implementation-testing cycles towards the end. Of course we have time left if something does not go as planned so we have time to react and work on any possible issues that we have to face.

Questions:

- Multiplayer is mentioned in the additional features. What does it mean in the context of tower defense?