Software Project Documentation,

Tower Defence

ELEC-A7151 - Object-oriented programming with C++

Nico Renlund 715874

Oskar Holmberg 706595

Sandra Schröder 713274

Victor Mesterton 652885

Martin Sederholm 716080

Table of Contents

1. Overview

The program works as a tower defence game. We have five different enemies that come in waves. Each wave gets harder and harder the further you come. The player can use six different towers to defend against the enemies, one of the towers is a utility tower. Some towers can only be placed on grass and some in water. The player has ten lives at the start of the game and if the player runs out of lives you lose. You lose a life when an enemy gets to the end of the path.

The features in the final product differ slightly from the plan. For example, we do not have any sound effects in our project. We also have more enemies and towers than we first planned. Moreover, towers can be added during the wave, but they are not

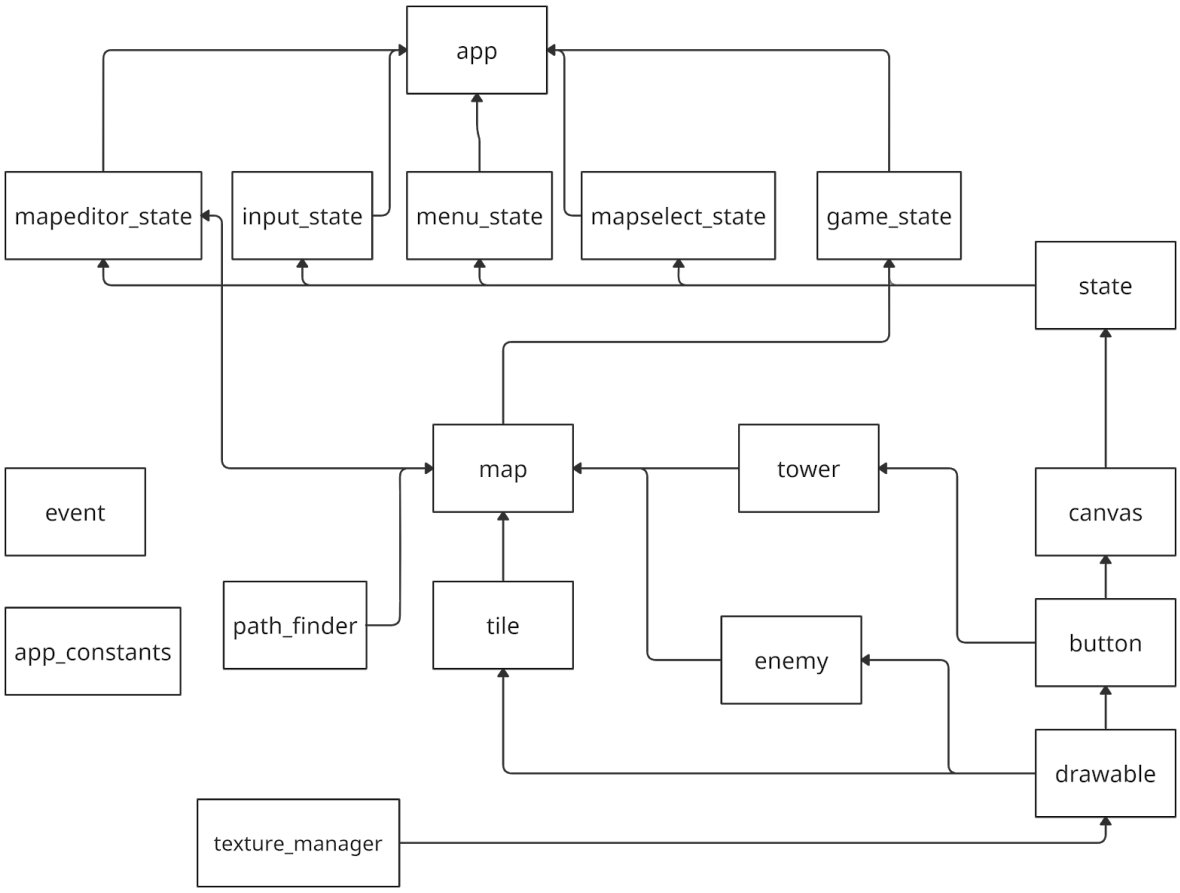
upgradable. The architecture got quite more complicated than expected due to some problems with SFML. However, we tried to keep it as simple as possible and think we managed to do that. We also needed to implement more classes for example for the canvases.

2. Software structure

The project is divided into 4 folders, where each one has their own function as follows:

- .vscode: VS-code settings
- doc: Documents related to the project
- plan: Initial project plan
- src: Game files. Class division is illustrated in the UML-diagram

As external libraries, we have used SFML, which is a graphical user interface, and C++ standard libraries.



The app controls which state is currently active. The state manages all the graphics in the window and responds to all user input, apart from the close window action, which the app controls. The map, which is used by both the map editor state and the game state, is a special kind of canvas that manages everything that can happen on a map, such as controlling enemy movement and towers. The map also holds a 2D vector of all the tiles and uses a pathfinder to direct the enemy movement, as well as validating a user made map in the map editor. A canvas is a part of the screen which can contain other canvases, buttons and drawables. This makes it easier to differentiate between things like the map and the sidebar menu.

Buttons and drawables are the core of the GUI. A button is a drawable that reacts to user input, and a drawable is a base class that needs to be inherited by anything that needs a texture. The drawable uses our texture manager to make sure we do not load the image files more than once. Enemies are derived from drawables as they do not need to react directly to any user input, while towers are derived from buttons because they do.

The events are the core of how communication works between different components, all the way from a state down to a button. The communication from the app to a state uses SFML events, while the state responds with our own events. Events can be used by anything apart from drawables (excluding classes derived from drawables).

App constants are all values and strings that will not change while the program is run but might get changed for balancing purposes. These can be used everywhere in the project.

3. Instructions for building and using the software

The project can be compiled in at least Linux and Windows (with MinGW). If SFML isn't installed by default it must be installed. All of the commands must be done from the src-folder. Building and compiling the project can be done with the command "make build". If "clean" is not working it must be done manually; first, run the command "make noclean" and after that run the command "rm *.o". This seemed to be an issue to some of us in Windows but has not happened in Linux. To run the project one should use different commands in Linux and Windows. In Linux the command is "make lrun" and on Windows "make wrun".

Illustration of the terminal output if "make clean" doesn't work:

```
PS C:\Users\KotiPC\Desktop\cpp_project\tower-defense-2020-5\src> make noclean
g++ -c *.cpp
g++ -c ./game/*.cpp
g++ -c ./gui/*.cpp
g++ -c ./utils/*.cpp
g++ -c ./map/*.cpp
g++ -c ./enemies/*.cpp
g++ -c ./towers/*.cpp
g++ *.o -o towerdefense -lsfml-graphics -lsfml-window -lsfml-system
PS C:\Users\KotiPC\Desktop\cpp_project\tower-defense-2020-5\src> rm *.o
PS C:\Users\KotiPC\Desktop\cpp_project\tower-defense-2020-5\src> make wrun
```

4. Testing

Our project has not been thoroughly tested using unit tests. However, it has been manually tested by running the application constantly, after each new implementation. This has worked relatively well as we started the construction of the project from the user interface. This has granted us the possibility of visually testing our application during the various construction practises.

During the development, we have used a testing mode that has allowed us to test various functionality in the user interface and game progress. This mode is available by simply using the test() function in main.cpp rather than app.run().

Functions that are not directly connected to the user interface, such as the PathFinder class, have been tested by firstly checking all the errors the IDE has given. Secondly, implementing a manual test to the program execution utilizing various print statements to the console, from where it is possible to determine the functionality of the class/function. After this stage, many functions have been further developed and we have added increased functionality, when it has been required by the development of the application, to ensure proper functionality.

In the late stages of development, we did balancing of the game by repeatedly running the game and evaluating its functionality. Between the recursions, we made small changes to the abilities of the towers, enemies and game progress, such as damage, health, cost, range and speed. This allowed us to develop a better performing game that is more progressive in difficulty. This stage has not been our main focus due to time constraints.

To conclude, we have been satisfied with our testing methods. Primarily because we have not gone through testing during the exercise rounds, and therefore have minimal knowledge about unit testing in c++. Thus, we have adopted a method that worked for us, which is described above. We have not encountered any grand problems that could have been solved more easily using, e.g., unit tests.

5. Work log

Week 1:

During week 1 all group members did the same activities. Therefore, we do not see any point in specifying each group member's activities.

Making of the project plan (3h) each. We had a project planning meeting where every group member participated. During the meeting, we constructed the project plan that was required for the project. Additionally, we made a rough plan on who would implement the different parts and what the final product would look like. We did not finish the plan during the first meeting. Thus, we organised a second meeting the following day, (1h) each, where we finalised our project plan. All group members participated in the second meeting as well.

Everyone’s vision of the final product was quite alike so it was easy to make the plan. The hardest part was probably to decide which external library we wanted to use. However, we concluded that SFML would be the best fit for our project.

Week 2:

During week 2 all group members did the same activities. Therefore, we do not see any point in specifying each group member's activities, except Martin.

Meeting with the group (3h). Some confusion among the group members because we hadn’t been contacted by our group advisor, about our kickoff meeting. Martin started to get familiar with SFML and made a Makefile.

Week 3:

During week 3 all group members did the same activities. Therefore, we do not see any point in specifying each group member's activities.

Again a few meetings with the group planning and getting to know SFML. Watching tutorials of SFML together and discussing the implementation and class divisions. This took an estimated time of 3 hours. We still had not heard anything from the course personnel. Thus, we continued to check both Teams and Emails if our group advisor had tried to make contact. Victor sent an email to the advisor (copy to the rest of the group) → no answer.

Week 4:

Still, no contact from the advisor so we started with the project by ourselves. Due to us working on the same files at the same time, we used LiveShare in VSCode. Because the LiveShare mode seems to mess with the version control in VSCode for everyone apart from the host, the majority of all commits are shown as if they were done by Martin, though the rest of the group also participated. Quite a lot of time went trying to get the program to compile on our windows machines, trying to figure out what programs we would need. Difficult to start before everything worked as expected.

Nico Renlund:
- Group meeting 5h
- Working on the enemies
- set up the programming environment
- Enemies

Sandra Schröder:
- Group meeting 5h
- Working on the towers
- set up the programming environment

Martin Sederholm:
- Group meeting 5h
- The base structure of the program
- Working on the map and texture manager
- Drawables

Oskar Holmberg:
- Group meeting 5h
- Working on tiles and texture manager

Victor Mesterton:
- Group meeting 5h
- Working on the pathfinder
- set up the programming environment

Week 5:
Still working a lot in LiveShare/Discord screen share.

Nico Renlund:
- Group meeting/work 15h
- Testing to push
- Enemies

Sandra Schröder:

- Group meeting/work 15h
- Teams message to contact the advisor → answer + booking time for a demo meeting
- Testing to push
- Towers

Martin Sederholm:

- Group meeting/work 30h
- Helping everywhere and everyone → leading the project
- Game architecture
- Canvas
- State
- Button
- Event

Oskar Holmberg:

- Group meeting/work 20h
- Fixing the map, map editor
- Adding constants
- States

Victor Mesterton:

- Group meeting/work 15h
- Fixing the map
- Implementing paths
- Textures

Week 6:

Still working a lot in LiveShare/Discord screen share.

Nico Renlund:

- Group meeting/work 30h
- Implementing more enemies + getting it to work with the other parts of the project
- Fixing bugs
- Documentation

Sandra Schröder:

- Group meeting/work 30h
- Fixing bugs
- Implementing more towers + getting them to work with the other parts of the project
- Documentation
- Testing to run the project on Aalto Linux
- Textures

Martin Sederholm:

- Group meeting/work 35h
- Still leading the project and helping other group members
- Fixing bugs
- Documentation
- Fixing the project to work on Linux

Oskar Holmberg:

- Group meeting/work 35h
- Fixing bugs
- Optimization
- Helping to implement more towers and enemies
- Documentation

Victor Mesterton:

- Group meeting/work 30h
- Fixing bugs
- Textures
- Game graphics
- Fixing moving + visual parts
- Documentation