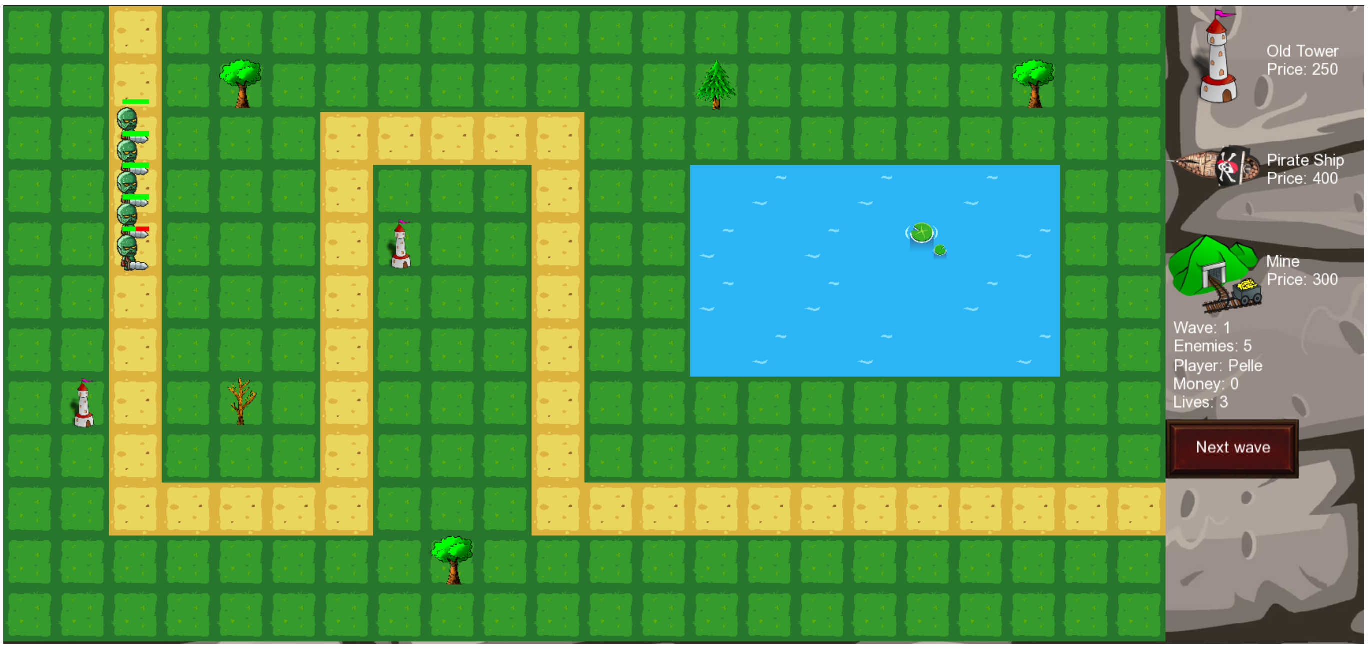
Tower Defence 1

Overview

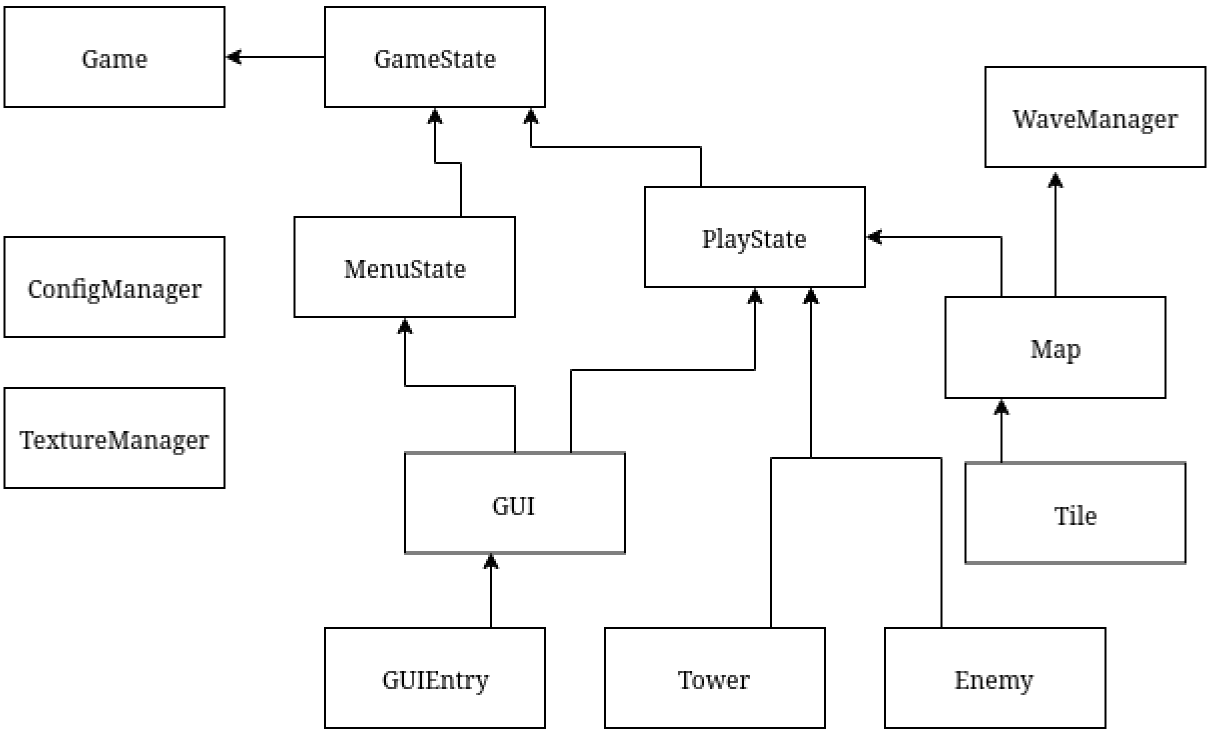
A fun medieval style tower defence game. Enemies come in waves and try to reach the player base, while the player builds towers to defend it. The game includes three different types of towers, five different types of enemies and over 20 waves. The game is lost when the player runs out of lives.



Wave: 1
Enemies: 5
Player: Pelle
Money: 0
Lives: 3

Software Structure

A rough, high level UML diagram of our game:



We use game states as a way of managing all the possible states our game uses, such as a main menu screen, an in-game state and a map selection screen. Our biggest class is by far PlayState. It is used to draw the map, the GUI and handles inputs. The PlayState provides most of the game functionality and is the state the game is in when the user is playing the game.

Our Game class handles the main drawing of the states. It fetches the first state from the stack and draws it onto the screen.

We have implemented different managers to handle different resources we might need. ConfigManager reads our settings file and handles the correct path to our map. WaveManager parses the waves.json file into a vector of different enemies for each wave. TextureManager is used to retrieve and load textures needed by other classes.

The Map class describes the in-game map and is used to load maps from text files. It also uses the WaveManager to load new waves. The Map consists of tiles which have different textures depending on the tile type. The Map class uses some functions from our namespace Pathfinder, which provides functionality for pathfinding. The pathfinder implements the A* search algorithm and calculates a path from the enemy spawn to the player's base, which the enemies then follow. The A* search algorithm is a bit overkill in our case but it would make it possible later to enable tower placing on road tiles for example.

The Tile class is rather simple. It just contains the texture name for the specific tile and also what type of tile it is.

Tower is an abstract class that has sub classes that represent different types of towers. The different towers have different functionalities such as a basic attacking tower, a ship tower that can only be placed on water tiles and a money mine tower that increases money obtained after a wave.

Enemies are made out of a single class, since all the enemies have the same functionality, so there is no need to make it abstract. Enemies have a type enumeration so that there's a way to distinguish different enemy types. The attributes of the enemies are defined in the .json files from which the waves are loaded.

We have Gui class which contains GuiEntries. The GuiEntries can have text or/and an image. We use the GuiEntries to simplify creating the GUI for the game since we easily can set the position of both the text and image and get the height/width of the GUI component in a nice way.

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

Prerequisites

Libraries needed for this project are listed below:

- SFML 2.4.2
- CMake >= 3.0
- Boost >= 1.65

SFML is a multi-platform and multi-language multimedia platform. This project utilizes SFML to draw the map tiles, enemies and towers. Please visit the official SFML website if you wish to install this to your system.

Aalto computers come pre-installed with SFML version 2.4, however, if you are using a newer version of SFML, you will have to downgrade your version for the game to run.

CMake is a multi-platform tool for building, testing, and packaging software. This project requires version 3.0 at minimum – the current version lies at 3.16. CMake can be found pre-installed on Aalto computers at version 3.10.2. We use CMake to make building the project a lot easier for anyone cloning the project.

Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing. This project requires version 1.65 at minimum, which is the version Aalto computers have.

Building the program

1. Add a ssh key to courses git settings. (Instructions on how to generate a new ssh key)
2. Clone the repository using: `git clone git@courses-git.comnet.aalto.fi:CPlusPlus/tower-defence-2019-1.git`
3. Run the `build.sh` script found in the main directory (`./build.sh`)
4. Run the `run.sh` script (`./run.sh`)
5. The executable can also be found under folder `/out` named *tower-defence*

Playing the Game

An example gif of playing the game can be seen in our README.md

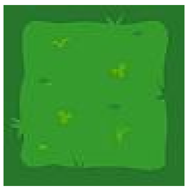
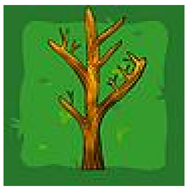
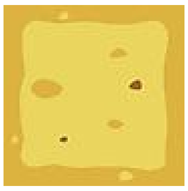
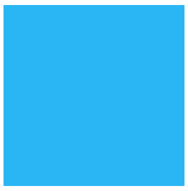
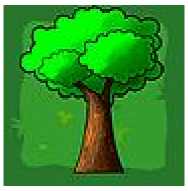
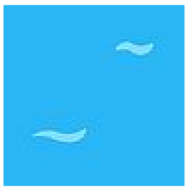
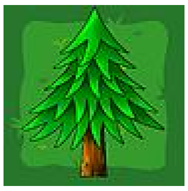
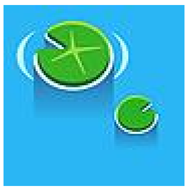
- Use the left mouse button to buy towers on the menu on the right side of the screen. You can place them on the map by left clicking on a tile.
- There are three different towers:
  - Old tower: A basic tower that can be placed on grass tiles. It shoots the enemy that is closest to the player base
  - Pirate ship: A ship tower that can only be placed on water tiles
  - Mine: A money generating, non-attacking tower that increases the amount of money obtained per wave
- If you click on a tower, a menu appears on the bottom where you can see its stats and upgrade/sell it. You also see the range of the tower as a transparent circle
- Press the M key to mute/umute the music
- Press + and - keys on the numpad to control volume

Creating your own maps and waves

Once you have tried the basic functionality of the game you could even try to make your own maps and waves!

See the maps/01 folder for an example on how to create a map. You can copy the files to a new folder 03 and then modify it as you want. The maptiles are defined in the map.txt file. The map has to be made rectangular, so make sure you don't have one row in the file that is longer or shorter than the others. See the table below for which character corresponds to which tile. The enemies must have a path that goes from the player base

(B) to the enemy spawn (S). You can't have more than one player base or enemy spawn.

Tile	Character	Tile	Character
	0		d
	#, B, S		W
	T		V
	t		w

The tile textures are taken from [incscope](https://inscope.itch.io/2d-tilemap). Enemy textures are from [craftpix](https://craftpix.net/freebies/free-monster-enemy-game-sprites/). Tower textures are from [opengameart](https://opengameart.org/).

To create your own waves, just modify the waves.json files. You can see an example of this in maps/01 folder.

Features implemented

In addition to all the basic features, we implemented the following more advanced features:

- **Non hardcoded maps (read from file)**
  - The maps are read from a file that will be in text format. The file will consist of a grid of symbols. Each symbol defines what each tile contains. The user is be able to resize the map to almost any size just by adding another row or column to the grid in the file.
- **Non hardcoded waves**
  - The waves are read from a json file and parsed using the boost library. This allows for easy wave editing as you can easily change how many enemies spawn in a certain wave, what type of enemies they are, their hp, speed etc.
- **Upgradeable towers**
  - The towers can be upgraded using the menu on the bottom. Each upgrade costs money and improves the tower's characteristics, for example range, damage, attack speed etc.
- **More kinds of enemies and towers**

  - We currently have five different enemy types and three different towers. The different enemies have different hp, speed etc. The different towers have different characteristics and functionality described in the playing the game section.
- **Dynamic enemy paths**
  - Dynamic enemy paths are be implemented using the A* search algorithm. More detailed information about this can be found in our project plan
- **Resizale window**
  - We made sure that the game window and all its elements are resizeable, so the player can choose how big a window they want to play on. This included making sure all the sprites scaled properly.
- **Background music**
  - The game has background music implemented using SFML's audio libraries. The sound files have to be in the open source .ogg-format to work.

Testing and Quality Assurance

The largest contributor to software quality was the extensive use of git branches and merge requests. We also made sure that our commit messages were clear and descriptive.

To test memory management and ensure its safety, valgrind was used to check for leaks. Valgrind was run with the following setup:`valgrind --leak-check=full --show-leak-kinds=all ./tower-defence`. Currently there might be some memory leaks found in the project but these don't stem from our code but rather from the different libraries used.

Testing was mostly done manually. Due to the graphical nature of the program and the scope of the project, testing manually was the only viable solution in many cases. For graphical interfaces manual testing was the most natural way to go; test and see if everything that we want is drawn correctly.

Other factors contributing to the software quality include programming parctices learnt during the course, such as smart pointers and the rule of five. Smart pointers were used in all places they were suitable and allowed. This ensured that we had no memory leaks when the pointer went out of scope.

Work log

Robin Karlsson

Responsibilities: Pathfinding, GUI

Week 45

- Group meeting 3h
- Game class and map drawing

Week 46

- Group meeting 3h
- Pathfinding

Week 47

- Group meeting 3h

- Towers and GUI

Week 48

- Group meeting 5h
- Texture management

Week 49

- Group meeting 5h
- GUI stuff
- Enemy spawning

Week 50

- Group meeting 5h
- Final touches
- Small fixes

Isac Sund

Responsibilities: Waves, Settings, GUI

Week 45

- Group meeting 3h
- General project setup 2h

Week 46

- Group meeting 3h
- Started implementing map loading 2h
- Fixing CMake issues 3h

Week 47

- Group meeting 3h
- Start implementing GUI with TGUI 3h

Week 48

- Group meeting 5h
- Remove TGUI due to bugs 3h
- Rewrite the game to use game states 5h

Week 49

- Group meeting 5h
- Implement GUI, enemies 6h
- Implement waves using json files

Week 50

- Group meeting 5h
- Implement map selection 3h
- Fix small things, bugs etc... 4h

Yann Jorelle

Responsibilities: Tower & map design, GUI

Week 45

- Group meeting 3h
- Initial setup, project plan etc. 3h

Week 46

- Group meeting 3h
- Tower and enemy classes 2h

Week 47

- Group meeting 3h
- Implement tower attacking 2h

Week 48

- Group meeting 5h
- Add basic enemy drawing 2h

Week 49

- Group meeting 5h
- GUI stuff 2h
- Map design and looking for textures 1h
- Add new tower type 2h
- Add music 0.5h

Week 50

- Group meeting 5h
- Add new tower type 1h
- Documentation 2h

Markus Henttonen

Responsibilities: Manager, random easy to implement stuff

Week 45

- Group meeting 3h

Week 46

- Group meeting 3h

- Converting old laptop to Linux-machine for more efficient project contribution 2h

Week 47

- Group meeting 3h
  - General structure for some classes

Week 48

- Group meeting 5h
-
  - Attempt salvage code from broken tower placement 1h

Week 49

- Group meeting 5h
- Familiarasing with other group members code 2h

Week 50

- Group meeting 5h
  - Gameplay balancing (income, tower upgrdes, wave spawner jsons), documentation
- Documentation 1h