Project documentation - Tower Defense

ELEC-A7151

Tower Defense Group 2:

Mimi Mokka, 652911

Anders Alho, 585075

Henry Pietilä, 597296

Jonna Mikkonen, 656289

Date August 28, 2020

Tower defense is a classic subgenre of strategy video game in which a player places towers on a game field to defend against enemies traversing through the field. The player aims to eliminate the enemies before they reach the other end of their path. If enough enemies reach the end of their path, e.g., the opposite end of the game field, the game is over. This documentation describes our version of a tower defence-game which we have implemented during the 2020 summer course of Object Oriented Programming with C++ as a course project. We have built a fully functioning tower defence game with a graphical user interface (GUI), and some additional features presented later in this documentation.

1 Overview

1.1 Scope of the work

We have implemented the basic features, including basic graphics, four different types of towers and enemies, and a simple user interface.

As required, we have implemented three types of towers: a basic tower which shoots enemies within its range, a slowing tower which slows down enemies within its range and a bomb tower which launches a bomb in the direction of an enemy in range. In addition, we have one extra tower, a super tower.

There are also different kinds of enemies whose properties differ from each other. Some enemies are stronger than others and some require more hits to kill. Some enemy types move faster and some will split into several smaller enemies when killed.

The game is played through simple graphic user interface (GUI) which shows information about the state of the game, such as money and lives, and the current game field with towers and enemies. The GUI is also used to buy, sell and place the towers, start rounds and pause the game.

1.2 Gameplay description

As the game starts, the player has some money to buy towers. The player can place the towers in the game field, and when they are done, a wave of enemies can be initialized. During the wave, towers shoot enemies inside their range. The player is free to buy and remove towers during the wave as well.

The enemies will follow a single, non-branched path. If an enemy reaches the end of the path, a life is lost. If the player runs out of lives, the game is lost.

After the wave the same logic repeats with increasing difficulty until the game is lost or all rounds are cleared. Increasing difficulty is implemented by increasing the speed and the amount of spawning enemies. In addition, on the first four levels the amount of different enemy types increases each level and at the last level only harder enemies are spawned.

The player gets money for each enemy killed, and said money can be used to by more towers during the game.

1.3 Additional features

From the start, the basic features were implemented in such a way that making additional features was straight-forward, requiring as little code refactoring as possible. We were able to implement a few additional features.

- More different kinds of enemies and towers
- Different attack and defense types for both the towers and the enemies
- Tower placement can be altered during enemy waves
- The player has lives and enemies reaching the base result in lives lost
- Enemy and Bullet animations

We first implemented three types of towers and enemies (described later in the documentation). Then we added one extra type for both classes. Our enemies respond differently towards different towers. For example, our SlowingTower slows down enemies differently.

Towers can be built and sold during enemy waves. The towers are intentionally designed not to be able to reposition themselves once built. This could be altered, if so desired, relatively easily by telling the towers stop instead of disallowing their moving.

Enemies have animations for movement and death. The bombs shot by the bomb tower have animation for the explosion. Animations are implemented using atlas sheets so that different pictures of entities are shown by showing different parts of a bigger picture rather than having to load a new picture from memory for every frame.

2 Software structure

2.1 High-level structure of the software

The main classes are Application, GameField, State, StateStack, SceneNode, Player, Tower, Enemy and Bullet. These classes run most of the game logic or are base classes to many important derived classes. We use Simple and Fast Multimedia Library SFML in our software, which gives us graphics, system and window handling functions and classes.

The Game Field stores the game field and handles most of the game logic that happens between towers, bullets and enemies. It regularly updates them and the sceneGraph which consists of all SceneNodes on the game field. Player class handles player input, such as mouse clicks. It stores some player resources, such as lives, money and game status. Class is also used in buying and selling towers. Other main classes are described in the following sections and figures.

2.2 Application, States and the StateStack

Application is the object that holds a StateStack and is responsible for updating, processing events and rendering the state to the SFML window. Therefore our application serves as a main interface to the SFML library. The relationship between the Application, StateStack and the different States is depicted in figure 1.

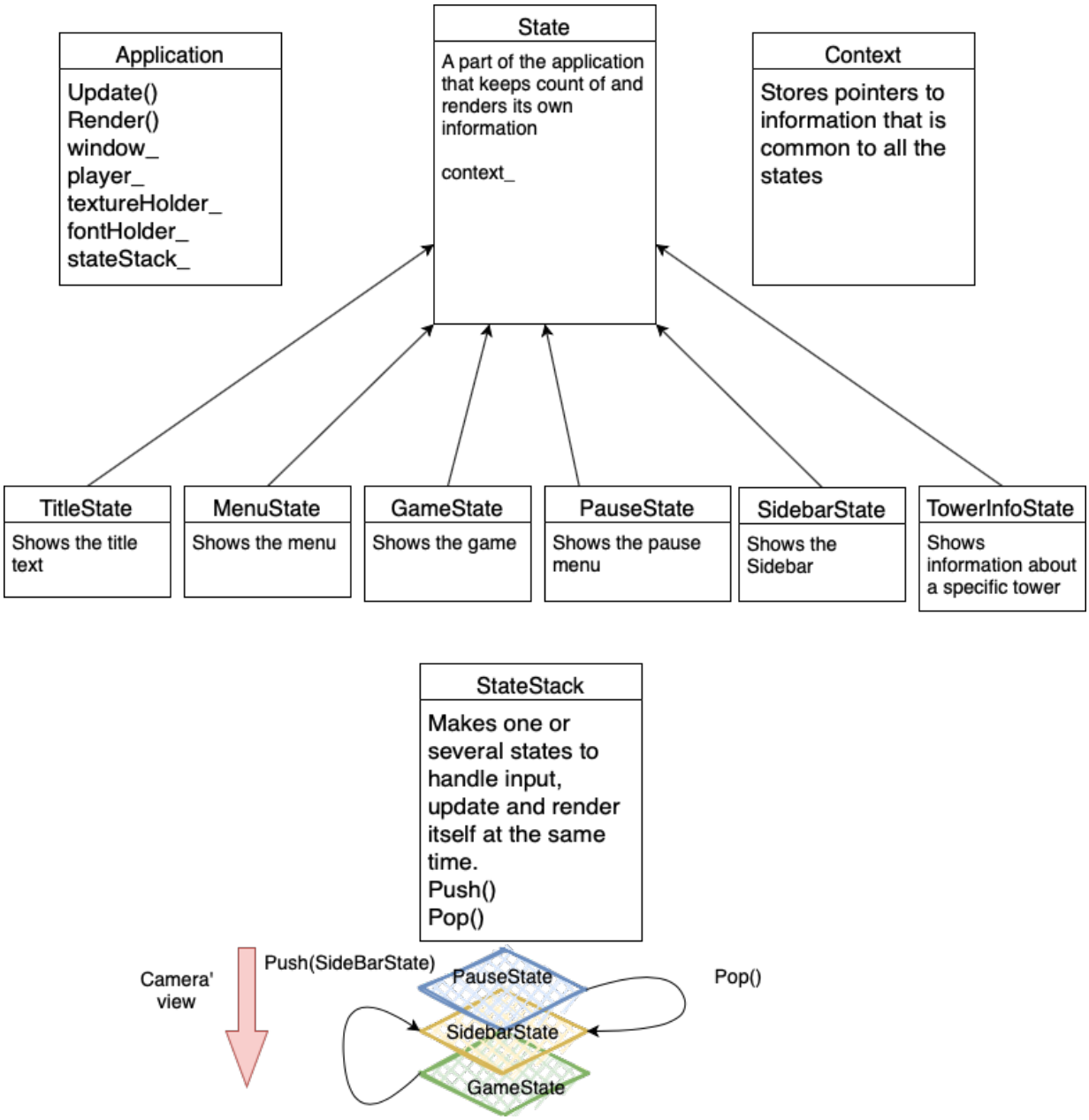

Figure 1: Diagram of high-level structure of the application

2.3 Enemies, Towers and Bullets

Enemies, Towers and Bullets are inherited from Entity class. Entity is a basic class for objects on the game field. Entity class two private attributes hitpoints_ and velocity_, which is a two dimensional vector. Entity class handles movement with SetVelocity function, and object's hitpoints with Damage and Destroy public member functions. Class has also getter functions GetVelocity and GetHitpoints to check these values. In addition, one can check if entity is alive or destroyed with IsDestroyed boolean function. Entity's protected UpdateCurrent function moves the Entity object on the game field according to its velocity. Entity class and its derived classes, Enemy, Tower and Bullet, are presented in figure 2.

We have an abstract enemy class that has the basic enemy attributes and functions. Enemy class inherits public and protected Entity member functions, so enemy class doesn't redefine functions for these features. Enemy class has functions to respond tower's attacks. Enemies can take hit from a bullet and take damage according to the bullet that hit it. Enemies can also be slowed down, so their speed is altered according to their slowDownRate. Player also gets money from destroying enemies, so each enemy knows how much money they are worth. Enemies have movement and death animations which are controlled in Enemy class. Some enemy types have also special behaviour when they are destroyed.

Different kinds of enemies are implemented in their own derived classes, which are presented in the picture 2. We have four different derived enemy classes for different enemy types, which are named according to enemy's qualities. All basic attributes of different types of enemies are presented in table 1. However all enemies have a difficultyLevel_ which increases as the level increases. This difficultyLevel_ is used to increase the speed of an enemy after each level. Speed increases by 25% in the beginning of each level after the first level is succesfully completed.

Basic enemies don't have anything too special about them, but they get destroyed immediately when they take hit and their slowDownRate is smaller than other enemies', so BasicEnemies speed slows down less than other one's. BasicEnemies are also first enemies to spawn in the game, and on the first level player tries to beat only basic enemies.

BulkEnemy has most hitpoints of all enemies. Because they are heavier than other enemies they also slow down more when they are in the range of SlowingTower. However BulkEnemies resist damage from basic an super bullets and take only half the damage when hit by these bullets.

MultiEnemy resists damage from BasicBullets taking only half the damage, but it takes double the damage from SuperBullets. In addition, if MultiEnemy is destroyed by a bomb, the amount of BasicEnemies that it spawns is doubled.

FastEnemy is twice as fast as other enemies so it is harder to hit than others. It resists damage from superBullets, taking only half the damage, but fast enemies take more damage from bombs by taking one and half times the bomb damage.

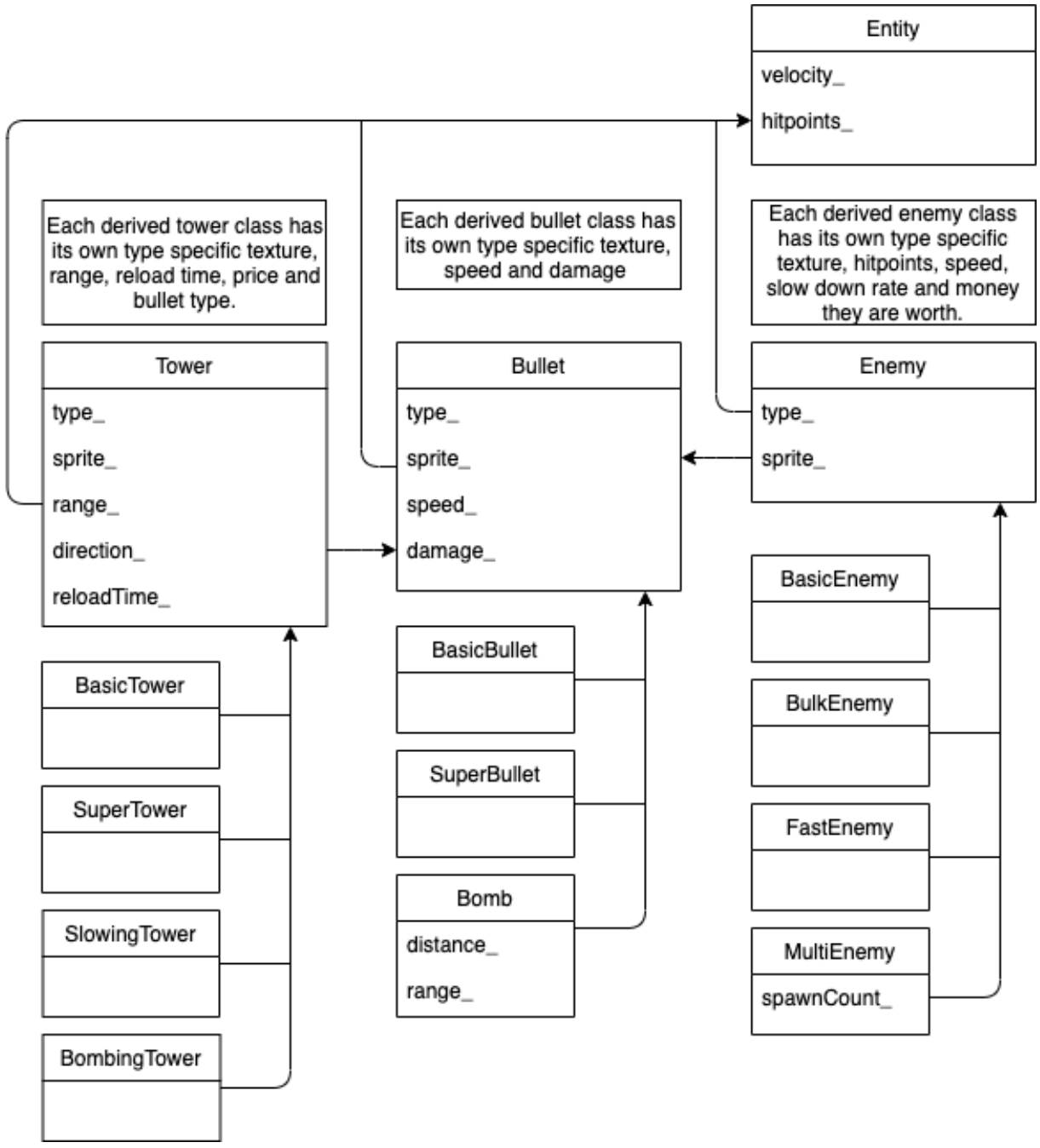

Figure 2: Diagram of Tower, Bullet and Enemy classes

Tower class is implemented similarly to Enemy class. We have an abstract Tower class with four different inherited classes. The inherited classes are depicted in table 2. Towers shoot bullet according to their type. An abstract Bullet class was implemented, along with three different inherited classes, depicted in table 3.

Enemies					
Type	Description	Hitpoints	Speed	Slow down rate	Worth of money
BasicEnemy	Resists slowing down	10	50	0.2	5
MultiEnemy	Multiplies into basic enemies upon killing	10	50	0.5	10
BulkEnemy	Takes more hits to kill	50	50	0.7	15
FastEnemy	Considerably faster that others	20	100	0.5	20

Table 1: Inherited Enemy classes

BasicTower and SuperTower shoot enemies within their range. They fundamentally have the same properties, but SuperTower has more effective destruction power, its range being bigger and reload time being smaller. SuperBullets are also faster than BasicBullets.

Slowing tower slows down enemies within its range, according to enemies' slow down rate. It does not shoot bullets nor can it destroy enemies, but it will make the job easier for other towers.

Bomb owers shoot Bombs in the direction of enemies within range. The Bomb will travel a distance given to it upon creation, and detonate at the end. All enemies within the range of the Bomb will take a hit.

Shooting towers rotate according to the direction they are shooting at. This direction is also given to the bullet upon creation. The time between shots is determined by reload time, in seconds.

2.4 Input handling and GUI

The input is handled by using the SFML-library's ability to handle input: sf::Event class. The actual logic is hidden inside the Player class. Player gives Commands to the GameField that relays the Commands to the game's Scene nodes. Scene nodes are things that can be transformed and drawn to the screen. This separation of input handling and game logic is depicted in Figure 3.

Towers					
Type	Description	Range	Reload time	Price	Bullet type
BasicTower	Shoots enemies within range	200	2	150	BasicBullet
SlowingTower	Slows down enemies within range	200	-	200	-
BombTower	Shoots bombs in the direction of enemies within range	150	3	350	Bomb
SuperTower	Shoots enemies within range	250	0.5	600	SuperBullet

Table 2: Inherited Tower classes

Bullets				
Type	Description	Speed	Damage	Range
BasicBullet	Causes damage to the enemy it hits	150	10	-
SuperBullet	Causes damage to the enemy it hits	200	5	-
Bomb	Causes damage to all enemies within range, when detonated	150	10	100

Table 3: Inherited Bullet classes

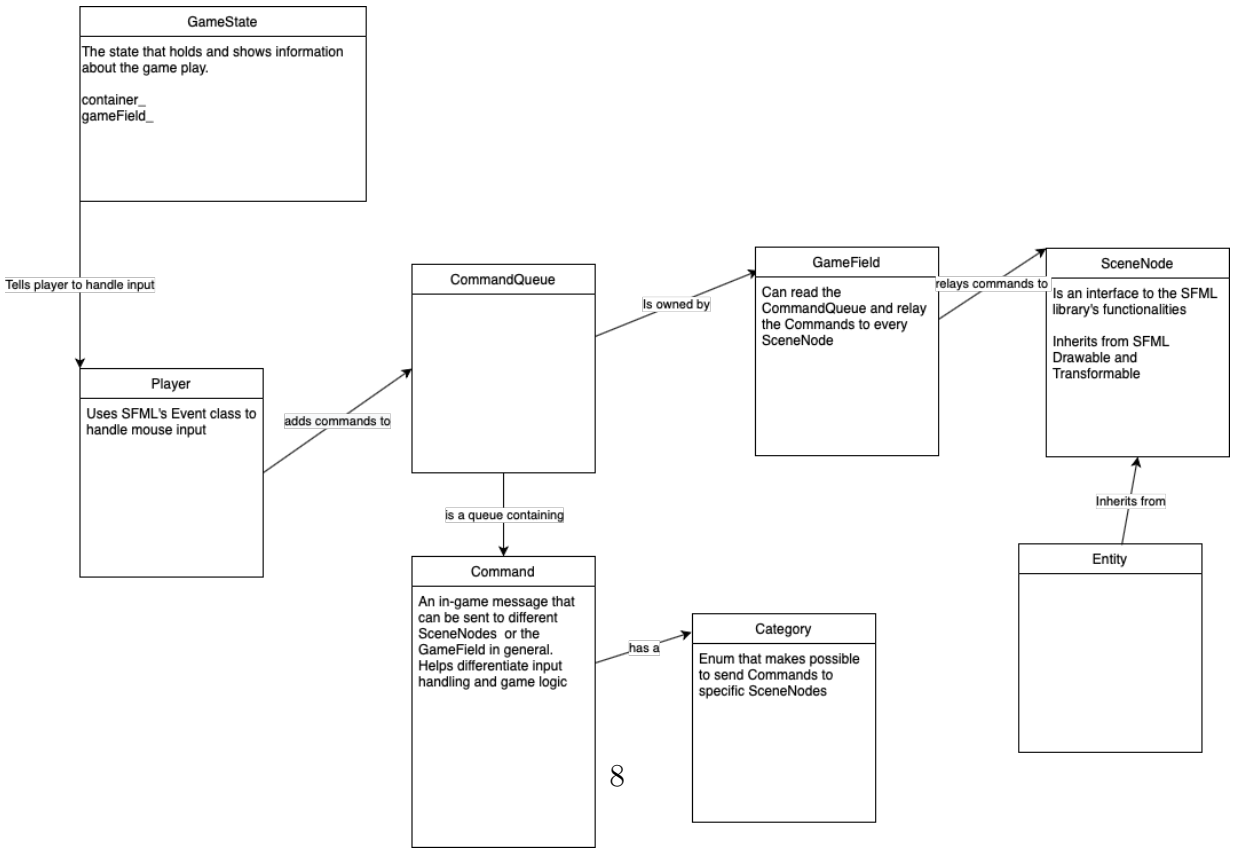

Figure 3: Structure of handling input

The sidebar is composed of two States: the SidebarState and the TowerInfoState. These two states represent the two states of the sidebar: the regular state and the inspect specific tower-state. The GUI is made with Components that may handle their own input. Buttons can send Commands via the Controller class that is used to relay the Commands (in-game event-related messages) to the GameField and the game entities. The sidebar states and the GUI Component hierarchy and relationships are shown in Figure 4.

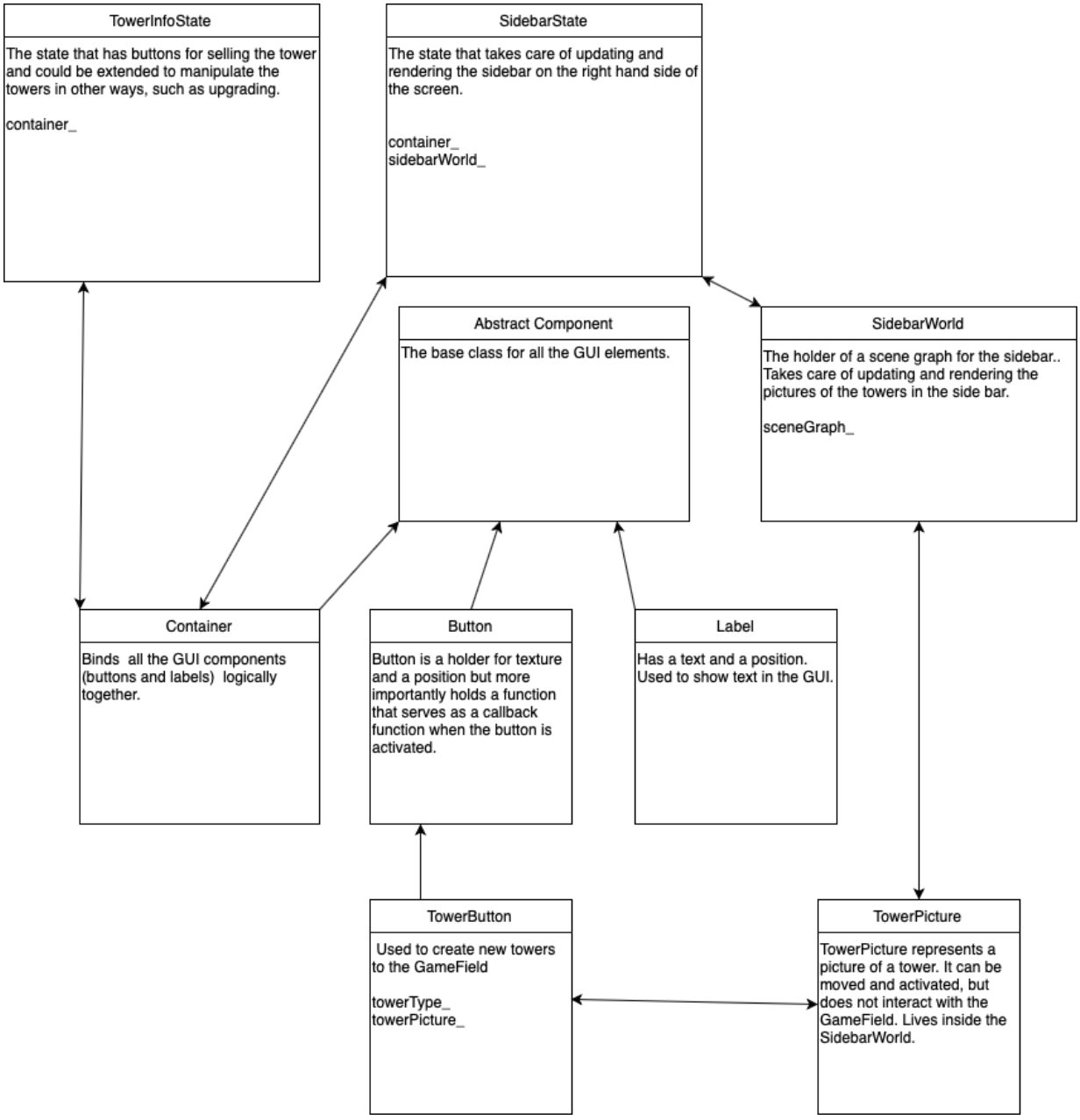

Figure 4: GUI class diagram

3 Instructions for building and using the software

To build the game, one should first create an empty directory for the build into the root of the repository:

```
mkdir build
```

Navigate to the directory created above:

```
cd build
```

Inside the build directory first run cmake and then make:

```
cmake ..
make
```

To run the game, run the executable farmitaistelu:

```
./farmitaistelu
```

3.1 Using the software

After compiling the program, a window is opened. When the game is started, we face a title menu, prompting the user to press any key. Then a main menu appears, giving the player an option between playing the game or quitting.

As the game starts, the player has some money to buy towers. The user can click on tower icons to buy them, move them to the field and then click again to place the tower on a chosen location. The circular range will turn red if the tower intersects with the enemy path or the sidebar, indicating that the user cannot place the tower here. The towers can be sold as well: if the user clicks on a tower located in the game field, the game prompts the user to choose between "Sell for x" or "Return".

At any point, the player can click on "Start level" and a wave of enemies is initialized. The enemies will follow a single, non-branched path. If an enemy reaches the end of the path, a life is lost. If the player runs out of lives, the game is lost. During the wave, towers shoot enemies inside their range. The player is free to buy and remove towers during the wave as well.

The game can be paused at any point, by clicking on the pause button at the left upper corner of the window. The player can then choose between returning to the main menu (this will result in the game being reset), quitting or continuing the game. The player can

also decide to retry the game at any point, by pressing the "Retry game" button, which will reset the game.

The game can be either lost or won. In either case, the user gets a message and an option between trying again, returning to the main menu or quitting the game.

The game can be relatively tricky. It can be won, for example, by following these instructions:

- Before the first level, buy a basic tower and a bomb tower and place them somewhere
- Run levels 1 and 2, these towers should be sufficient to pass
- In the beginning of the third wave, buy a slowing tower and place it near bomb tower. When you have enough money buy another bomb tower. You can also sell the basic tower to buy the second bomb tower
- Buy another slowing tower during the fourth wave and place it near the second bomb tower
- When you have enough money, buy a super tower. Do this preferably before you start the fifth level.
- During the fifth level buy first slowing tower and place it near super tower. This should be enough, but you can buy more towers as you get money from the enemies destroyed.

3.2 Libraries

The game uses SFML graphics library. We chose it beacause it seemed beginner friendly yet sufficient for our needs. No other libraries were needed apart from the standard libraries.

4 Testing

All parts of software were carefully tested when implemented by first printing all relevant information to the console and then by playing the game in different kinds of ways. We also utilized SFML's assert function, which allows easier debugging for problems and bugs which might be hard to find and fix later on.

We didn't use any automated testing although we planned to do it at first. None of us were familiar making automated tests and it seemed it would have required a lot of time and effort to make tests which would have tested the game throughout even though it is quite simple.

5 Work log

We created a Telegram group in the beginning of the project, where we have discussed the work and shared ideas. We have met multiple times via video call (Google Hangouts) and a few times in person, all though not so that all group members have been present.

We also created a Sheets file in Google Drive, where we marked each member's weekly working hours. The sheet included multiple categories: Planning, Enemy, Game, Gamefield, Tower, Design, Tools, Libraries, Graphics and Documentation.

The work division ended up so that Jonna was mainly responsible for the Enemy class and Mimi for the Tower class. Bullet class implementation was divided between them, as the classes intersect a lot. Anders developed the GUI and underlying logic and also most of the GameField and Application classes with Henry. Henry has also been responsible for the graphics and animations.

As said, Jonna was responsible of all enemy classes and was also mainly responsible in making collision logic and detection between bullets and enemies. She also did some states, for example GameOverState, and different game logic related things when needed.

Mimi was responsible for the Tower class, though many features were implemented also by others. She developed the Bullet class and towers' shooting feature. She was also responsible for inherited tower and bullet classes, including Bomb and Bomb tower classes. In addition she developed part of the GameField class.

Anders and Henry did most of the application logic, such as event and input handling and game states, GUI and graphics. They worked quite a lot together and were responsible for developing major application logic parts and user interfaces. They also ended up fixing probably most of the bugs we detected.

Table 4 describes roughly what was done each week and how many hours each group member used. Summed up, Mimi used in 83, Henry 90, Anders 96 and Jonna 98 hours in total. In short, we think all group members contributed roughly the same amount, and we are content with how the work load was divided.

Work log		
Week 1	We spent the week learning SFML We made a blob on the screen with SFML Some basic implementation for first classes and graphics were made	Mimi 0 Henry 10 Anders 10 Jonna 8
Week 2	Further planning and studying of libraries	Mimi 20 Henry 8 Anders 8 Jonna 12
Week 3	Graphics and design took a huge step forwards Enemy and tower classes were implemented and made functioning Enemy path was implemented, so were movable towers Towers started shooting bullets Different state classes were implemented, including main menu Mid-term meeting with the assistant	Mimi 15 Henry 32 Anders 32 Jonna 20
Week 4	Enemy class developed, different inherited classes implemented Sprites and animation for enemies Towers now shoot better, they can aim and range is working Slowing towers implemented Sidebar for buying towers and operating the game	Mimi 14 Henry 2 Anders 4 Jonna 16
Week 5	Bombs and Bomb towers implemented States improved, the game can be lost or won Documentation was started Money system implemented, towers can be bought	Mimi 8 Henry 11 Anders 10 Jonna 14
Week 6	Commenting code Refining game logic Final touches and project demonstration Final bug fixes and documentation Deadline	Mimi 26 Henry 27 Anders 32 Jonna 28

Table 4: Description of what was done every week and roughly how many hours were used, for each project member