

## Orcs n Towers

Generated by Doxygen 1.9.5



<b>1 Orcs n Towers</b>	<b>1</b>
1.1 Overview	1
1.2 Instructions	1
1.3 How to compile the program	2
1.4 Testing	2
1.5 Work log	3
1.6 Software structure	8
<b>2 Source content</b>	<b>9</b>
<b>3 Hierarchical Index</b>	<b>11</b>
3.1 Class Hierarchy	11
<b>4 Class Index</b>	<b>13</b>
4.1 Class List	13
<b>5 File Index</b>	<b>15</b>
5.1 File List	15
<b>6 Class Documentation</b>	<b>17</b>
6.1 BombProjectile Class Reference	17
6.2 BombTower Class Reference	17
6.2.1 Detailed Description	17
6.2.2 Constructor & Destructor Documentation	18
6.2.2.1 BombTower()	18
6.2.3 Member Function Documentation	18
6.2.3.1 shoot()	18
6.2.3.2 update()	18
6.3 BulletProjectile Class Reference	19
6.3.1 Detailed Description	19
6.3.2 Member Function Documentation	19
6.3.2.1 hasHitEnemy()	19
6.3.2.2 textureType()	20
6.3.2.3 update()	20
6.4 BulletTower Class Reference	20
6.4.1 Detailed Description	21
6.4.2 Constructor & Destructor Documentation	21
6.4.2.1 BulletTower()	21
6.4.3 Member Function Documentation	21
6.4.3.1 shoot()	21
6.5 Button Class Reference	22
6.5.1 Detailed Description	22
6.5.2 Constructor & Destructor Documentation	22
6.5.2.1 Button()	22

6.5.3 Member Function Documentation	23
6.5.3.1 isClicked()	23
6.6 Enemy Class Reference	23
6.6.1 Constructor & Destructor Documentation	24
6.6.1.1 Enemy()	25
6.6.2 Member Function Documentation	25
6.6.2.1 dead()	25
6.6.2.2 getCenter()	25
6.6.2.3 getHealthText()	26
6.6.2.4 getLocation()	26
6.6.2.5 getMoney()	26
6.6.2.6 getWaypoints()	26
6.6.2.7 hp()	26
6.6.2.8 initialHp()	27
6.6.2.9 isWaypointPassed()	27
6.6.2.10 poisonStatus()	27
6.6.2.11 setVelocity()	27
6.6.2.12 slowedStatus()	27
6.6.2.13 speed()	28
6.6.2.14 type()	28
6.6.2.15 update()	28
6.7 Explosion Class Reference	28
6.7.1 Detailed Description	29
6.7.2 Constructor & Destructor Documentation	29
6.7.2.1 Explosion()	29
6.7.3 Member Function Documentation	29
6.7.3.1 update()	29
6.8 FreezingTower Class Reference	30
6.8.1 Detailed Description	30
6.8.2 Constructor & Destructor Documentation	30
6.8.2.1 FreezingTower()	30
6.8.3 Member Function Documentation	31
6.8.3.1 shoot()	31
6.8.3.2 update()	31
6.8.3.3 upgradeTower()	32
6.8.4 Member Data Documentation	32
6.8.4.1 lockedEnemies_	32
6.8.4.2 slowCoefficient_	32
6.9 Game Class Reference	32
6.9.1 Detailed Description	34
6.9.2 Member Function Documentation	34
6.9.2.1 checkTowers()	34

6.9.2.2 <code>getPath()</code>	35
6.9.2.3 <code>processEvents()</code>	35
6.9.2.4 <code>render()</code>	35
6.9.2.5 <code>run()</code>	35
6.9.2.6 <code>update()</code>	36
6.9.2.7 <code>updateMenus()</code>	36
6.10 LevelManager Class Reference	36
6.10.1 Detailed Description	37
6.10.2 Constructor & Destructor Documentation	37
6.10.2.1 <code>LevelManager()</code>	37
6.10.3 Member Function Documentation	37
6.10.3.1 <code>getCurrentLevel()</code>	38
6.10.3.2 <code>getLevelTotal()</code>	38
6.10.3.3 <code>readingSuccessful()</code>	38
6.10.3.4 <code>readLevels()</code>	38
6.10.3.5 <code>update()</code>	39
6.10.4 Member Data Documentation	39
6.10.4.1 <code>levelSpecs_</code>	39
6.11 Map Class Reference	39
6.12 Menu Class Reference	40
6.12.1 Detailed Description	41
6.12.2 Member Function Documentation	41
6.12.2.1 <code>canBePlaced()</code>	41
6.12.2.2 <code>checkButtons()</code>	41
6.12.2.3 <code>createMenu()</code>	42
6.12.2.4 <code>drag()</code>	42
6.12.2.5 <code>draw()</code>	42
6.12.2.6 <code>drawRange()</code>	43
6.12.2.7 <code>newTower()</code>	43
6.12.2.8 <code>update()</code>	43
6.13 MissileProjectile Class Reference	43
6.13.1 Detailed Description	44
6.13.2 Constructor & Destructor Documentation	44
6.13.2.1 <code>MissileProjectile()</code>	44
6.13.3 Member Function Documentation	44
6.13.3.1 <code>hasHitEnemy()</code>	45
6.13.3.2 <code>textureType()</code>	45
6.13.3.3 <code>update()</code>	45
6.14 MissileTower Class Reference	46
6.14.1 Detailed Description	46
6.14.2 Constructor & Destructor Documentation	46
6.14.2.1 <code>MissileTower()</code>	46

---

6.14.3 Member Function Documentation	47
6.14.3.1 shoot()	47
6.15 path Class Reference	47
6.15.1 Constructor & Destructor Documentation	48
6.15.1.1 path()	48
6.15.2 Member Function Documentation	48
6.15.2.1 addWaypoint()	48
6.15.2.2 readingSuccessfull()	48
6.15.2.3 readPath()	49
6.16 Player Class Reference	49
6.16.1 Detailed Description	49
6.16.2 Constructor & Destructor Documentation	50
6.16.2.1 Player()	50
6.16.3 Member Function Documentation	50
6.16.3.1 addMoney()	50
6.16.3.2 getHP()	50
6.16.3.3 getLevel()	51
6.16.3.4 getWallet()	51
6.16.3.5 removeHP()	51
6.16.3.6 removeMoney()	51
6.17 PoisonTower Class Reference	52
6.17.1 Detailed Description	52
6.17.2 Constructor & Destructor Documentation	52
6.17.2.1 PoisonTower()	52
6.17.3 Member Function Documentation	53
6.17.3.1 shoot()	53
6.17.3.2 update()	53
6.17.4 Member Data Documentation	53
6.17.4.1 lockedEnemies_	54
6.18 Projectile Class Reference	54
6.18.1 Detailed Description	55
6.18.2 Constructor & Destructor Documentation	55
6.18.2.1 Projectile()	55
6.18.3 Member Function Documentation	55
6.18.3.1 destroy()	55
6.18.3.2 distToTower()	56
6.18.3.3 getDamage()	56
6.18.3.4 getShootDir()	56
6.18.3.5 getSpeed()	56
6.18.3.6 getType()	56
6.18.3.7 hasHitEnemy()	57
6.18.3.8 textureType()	57

---

6.18.3.9 update()	57
6.19 ResourceContainer< T_enum, T_resource > Class Template Reference	57
6.19.1 Detailed Description	58
6.19.2 Member Function Documentation	58
6.19.2.1 get()	58
6.19.2.2 load()	58
6.20 Tower Class Reference	59
6.20.1 Detailed Description	60
6.20.2 Constructor & Destructor Documentation	60
6.20.2.1 Tower()	60
6.20.3 Member Function Documentation	61
6.20.3.1 enemyWithinRange()	61
6.20.3.2 shoot()	61
6.20.3.3 update()	61
6.20.3.4 updateFireTimer()	62
6.20.3.5 upgradeTower()	62
6.20.4 Member Data Documentation	62
6.20.4.1 type_	62
<b>7 File Documentation</b>	<b>65</b>
7.1 bombProjectile.hpp	65
7.2 bombTower.hpp	65
7.3 bulletProjectile.hpp	65
7.4 bulletTower.hpp	66
7.5 button.hpp	66
7.6 enemy.hpp	66
7.7 explosion.hpp	68
7.8 freezingTower.hpp	68
7.9 game.hpp	69
7.10 levelManager.hpp	70
7.11 map.hpp	71
7.12 menu.hpp	71
7.13 missileProjectile.hpp	72
7.14 missileTower.hpp	72
7.15 path.hpp	72
7.16 player.hpp	73
7.17 poisonTower.hpp	74
7.18 projectile.hpp	74
7.19 resource_container.hpp	75
7.20 tower.hpp	75
<b>Index</b>	<b>77</b>





# Chapter 1

## Orcs n Towers

### 1.1 Overview

Orcs n Towers is a tower defense game set in a fantasy setting, where orcs and other such monsters try to reach and destroy the player's castle, the player must defend against the monsters by placing different towers with specific roles. The player will have a set number of hitpoints that are depleted when enemies reach the castle. When all hitpoints are lost the player loses.

The monsters traverse a path, along which the player can place their towers. There is three different paths of which one is chosen at random for the duration of the game. Once a monster is inside a towers range, depending on the towers it will either create a projectile that matches the towers type, or apply a slowing or poison effect on the monster. Certain towers can only affect certain monsters.

The player can buy as many Towers as they can afford throughout the game, as well as upgrade them to increase the damage the tower will cause the monster and sell them. The player earns money by killing enemies as well as by progressing through the levels.

The game has 5 different levels of increasing difficulty, by introducing more monsters in amount and type at quicker intervals. The game is won once the player has defeated all levels. The player loses HP every time a monster reaches the castle, and once the HP is zero, the game is lost.

### 1.2 Instructions

Once you have started the game, to place towers on map, drag and drop them from the side bar to an appropriate place. Note that towers cannot be built on the path. To cancel the purchase of a tower drag it back onto the side bar. The towers range can be seen while dragging it as well as by clicking on it once on map. To upgrade or sell a tower, click on the tower and choose the wanted action from the menu that appeared on the bottom of the screen, there you can also see the towers specifications.

A level is completed once all enemies from that level have been killed. To move on to the next level, press the "next level" button that appears on the screen. The game can be paused by pressing the "pause" button on the side bar, there the player can also see their current level and how much money and HP they have.

Custom levels and paths can be created in levels.csv and paths.csv respectively, found in assets folder, read formatting instructions carefully.

## 1.3 How to compile the program

To compile the game, as taken from git, on the command line:

```
1. create an empty directory where the build files will be written
2. change directory to that directory
3. run: cmake ..
4. run: make
5. run: ./TD
```

SFML multi-media library (minimum version 2.5) is required.

## 1.4 Testing

Testing was mostly done directly in the source files, in either a project branch or on master branch. The first kinds of tests were simply rendering the game objects to be able to see them on the screen, once that was at least partially working, it was easier to gauge what exactly the game objects were doing and testing how the objects interacted with each other was started. Print statements were used as well to make it easier to follow which part of the code was being executed, and if it was the expected part.

Enemies movement was initially tested by hardcoding waypoints, to see that the logic worked, and enemies traversed the path that they were supposed to. Once towers and enemies were able to be rendered on the screen, their interactions with each other could be tested, namely that towers recognized that enemies were within their range and could pick one to target.

When towers could lock on to enemies, the creation of projectiles by towers could be tested. Initially there were some minor issues with initializing projectiles due to different ideas on what should be passed to constructor, but that was easily solved with some adjustments. Once projectiles could be created their movement and ability to hit enemies was tested, initially they didn't seem to move; with some adjustments to their values that dictated how far they could move from their tower, it could be determined that projectiles were able to move towards enemies and hit them, and therefore cause damage to them.

When projectiles could hit enemies, the killing of enemies could be better tested, to see that enemies would actually take damage from projectiles, and once their HP would reach zero, they would die and be deleted, which they did. The testing of enemies causing damage to the player by reaching the castle, and dying when they do so, was done by allowing the enemies to reach the castle.

Not being able to buy towers if player didn't have enough money or, place towers on top of each other or on the path was simply tested by trying to do so. User interactions with the game, like the ability to pause/unpause, displaying tower information, upgrading or selling towers and moving on to the next level were tested by executing the action and observing the outcome.

Reading both levels and paths from file was tested by reading the content into containers and printing the contents as well as the status of the reading success. Firstly with correctly formatted input to see that the reading logic worked, and then with incorrectly formatted input, to test the error handling. As expected, incorrectly formatted input caused reading success to return false, and thus indicating reading failed. Once it was determined the reading of levels worked, the level execution was tested by playing through the whole game.

Additionally, we had encountered segmentation faults on different stages of development. These were addressed by running the executable with GNU Debugger.

## 1.5 Work log

### 1.5.0.0.1 Division of work / main responsibilities: Pavel Filippov:

- [Tower](#) class and it's derived classes (bullet-, bomb-, missile-, poison-, and freezing tower)
- [Game](#) class

Otto Litkey:

- Graphics (buttons, textures)
- User interaction
- [Menu](#) class
- Resource container template class

Ellen Molin:

- [Projectile](#) class and it's derived classes (bullet-, bomb-, and missile projectile)
- [LevelManager](#) class
- Reading paths from file
- [Player](#) class

Leo Saied-Ahmad:

- [Enemy](#) class

Tuan Vu:

- [Map](#) class
- Path class
- Some graphics related to map and path of the game.

**1.5.0.0.2 Weekly breakdown Week 1**

Pavel Filippov:

- Initialised implementation of base tower class.
- Estimated workload: 10 hours

Otto Litkey:

- Initialised implementation of class(es) responsible for graphics.
- Estimated workload: 6 h

Ellen Molin:

- Initialised implementation of [Player](#) and base [Projectile](#) classes.
- Estimated workload: 5h

Leo Saied-Ahmad:

- Initialised implementation of base [Enemy](#) class.
- Estimated workload:

Tuan Vu:

- Initialised implementation of [Map](#) class.
- Estimated workload: 6h

**Week 2**

Pavel Filippov:

- Continued implementation of base tower class.
- Estimated workload: 10 hours

Otto Litkey:

- Initialised game class and [ResourceContainer](#) template class.
- Estimated workload: 7 h

Ellen Molin:

- Continued implementation of [Player](#) and base [Projectile](#) classes.
- Estimated workload: 8h

Leo Saied-Ahmad:

- Continued implementation of base [Enemy](#) class.
- Estimated workload:

Tuan Vu:

- Continued implementation of [Map](#) class.
- Estimated workload: 8h

### Week 3

Pavel Filippov:

- Implemented update function for game class, as well as for towers.
- Estimated workload: 10 h

Otto Litkey:

- Tested rendering, beginnings of dragging and dropping functionality for creating towers.
- Estimated workload: 10 h

Ellen Molin:

- Improved projectile class and added functionality.
- Estimated workload: 10h

Leo Saied-Ahmad:

- Improved enemy class functionality, specifically kill and death functions.
- Estimated workload:

Tuan Vu:

- Finished implementing loading map from file, worked on drawing and being able to sell towers
- Estimated workload: 10h

### Week 4

Pavel Filippov:

- Created derived classes `bulletTower` and `bombTower` from `tower`. Moved tower update logic to it's own function.
- Estimated workload: 15 h

Otto Litkey:

- Continued testing rendering, finished drag and drop functionality for creating towers, and added way to pause game.
- Estimated workload: 10 h

Ellen Molin:

- Created derived classes bullet and bomb from projectile. Improved projectile methods to better work with derived classes.
- Estimated workload: 12h

Leo Saied-Ahmad:

- Created path class for directing enemy movement, and updated enemy's move function to make use of it.
- Estimated workload:

Tuan Vu:

- Continued working on map class so that it worked well with other classes.
- Estimated workload: 10h

## **Week 5**

Pavel Filippov:

- Implemented missile tower. Worked on rendering projectiles.
- Estimated workload: 15 h

Otto Litkey:

- Implemented a level system in player class. Created menu class, migrated UI elements to work from here. Worked on rendering projectiles.
- Estimated workload: 15 h

Ellen Molin:

- Created a missile projectile that follows enemy.
- Estimated workload: 12h

Leo Saied-Ahmad:

- Worked on enemies that split when hit. Worked on graphics: show players state, end screen when player loses.
- Estimated workload:

Tuan Vu:

- Worked on map class, loading background from file
- Estimated workload: 10h

## Week 6

Pavel Filippov:

- Implemented freezing effect tower and poison effect tower. Refined tower logic.
- Estimated workload: 15 h

Otto Litkey:

- Implemented explosions class to visualise bombs' explosions. Worked on graphics: created textures, show tower ranges. Improved logic behind user interactions with game objects.
- Estimated workload: 12 h

Ellen Molin:

- Implemented a levelManager that handles creating and managing levels, reads from file. Added functionality to load paths from file.
- Estimated workload: 15h

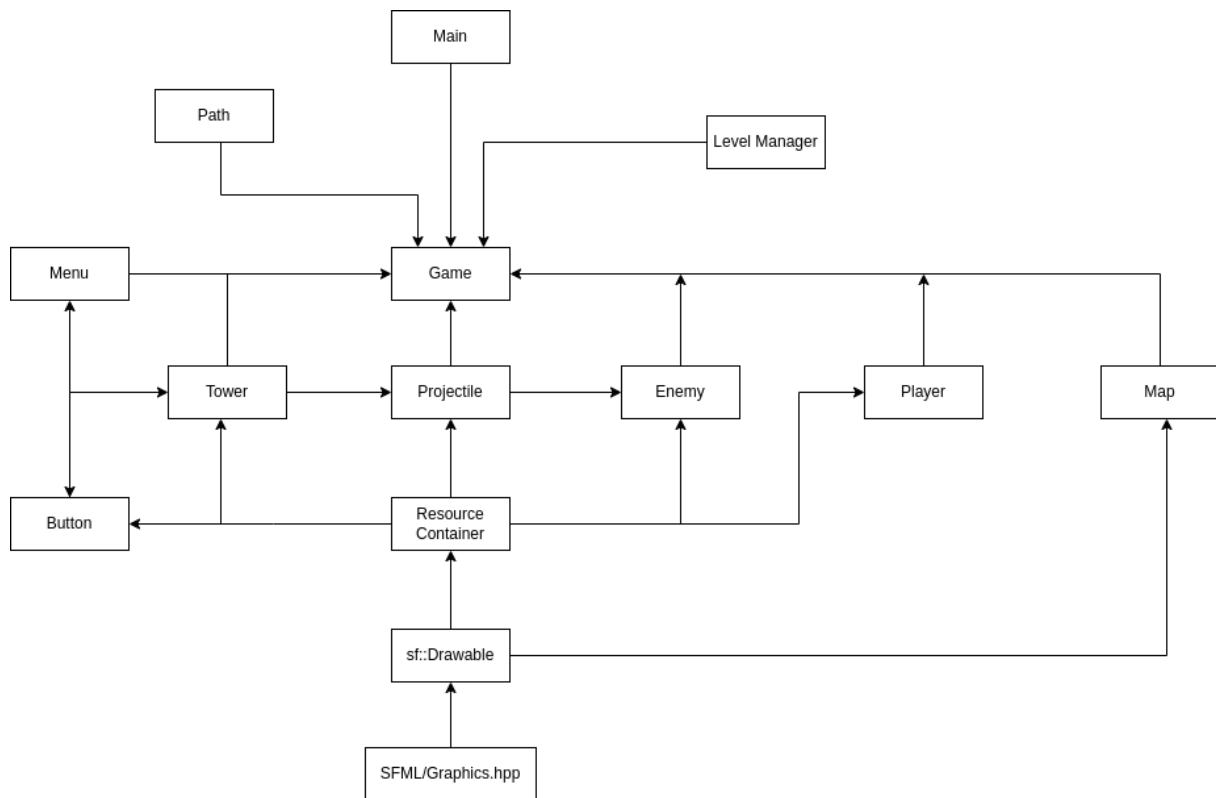
Leo Saied-Ahmad:

- Implemented slowing effect on enemies and refined enemy movement. Added health status over enemies.
- Estimated workload:

Tuan Vu:

- Implemented a path class that facilitates the creation of paths, incorporating functionality to prevent towers from being built on the designated path.
- Estimated workload: 10h

## 1.6 Software structure



The above image describes the relations between the main classes in the program.



## Chapter 2

### Source content

This folder should contain only `hpp/cpp` files of your implementation. You can also place `hpp` files in a separate directory `include`.

You can create a summary of files here. It might be useful to describe file relations, and brief summary of their content.



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

sf::CircleShape	
Explosion	28
sf::Drawable	
Map	39
Game	32
LevelManager	36
Menu	40
path	47
ResourceContainer< T_enum, T_resource >	57
ResourceContainer< Textures::EnemyID, sf::Texture >	57
ResourceContainer< Textures::ProjectileID, sf::Texture >	57
ResourceContainer< Textures::TowerID, sf::Texture >	57
ResourceContainer< Textures::Various, sf::Texture >	57
sf::Sprite	
Button	22
Enemy	23
Player	49
Projectile	54
BombProjectile	17
BulletProjectile	19
MissileProjectile	43
Tower	59
BombTower	17
BulletTower	20
FreezingTower	30
MissileTower	46
PoisonTower	52
sf::Transformable	
Map	39



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">BombProjectile</a>		
	<a href="#">Projectile</a> that causes damage to multiple enemies . . . . .	17
<a href="#">BombTower</a>		
	Represents the <a href="#">BombTower</a> class . . . . .	17
<a href="#">BulletProjectile</a>		
	<a href="#">Projectile</a> that travels in a straight line and can hit only one enemy . . . . .	19
<a href="#">BulletTower</a>		
	Represents the <a href="#">BulletTower</a> class . . . . .	20
<a href="#">Button</a>		
	Represents a clickable button . . . . .	22
<a href="#">Enemy</a>		23
<a href="#">Explosion</a>		
	Small class for drawing bomb explosions . . . . .	28
<a href="#">FreezingTower</a>		
	Represents the Freezing <a href="#">Tower</a> class . . . . .	30
<a href="#">Game</a>		
	This class runs the game logic . . . . .	32
<a href="#">LevelManager</a>		
	Handles the creation and managing of levels . . . . .	36
<a href="#">Map</a>		39
<a href="#">Menu</a>		
	Class for storing a collection of buttons, a menu . . . . .	40
<a href="#">MissileProjectile</a>		
	A projectile that targets (follows) a specific enemy . . . . .	43
<a href="#">MissileTower</a>		
	Represents the <a href="#">MissileTower</a> class . . . . .	46
<a href="#">path</a>		47
<a href="#">Player</a>		
	Class representing the player . . . . .	49
<a href="#">PoisonTower</a>		
	Represents the Poison <a href="#">Tower</a> class . . . . .	52
<a href="#">Projectile</a>		54
<a href="#">ResourceContainer&lt; T_enum, T_resource &gt;</a>		
	Template container for textures etc resources . . . . .	57
<a href="#">Tower</a>		
	Represents abstract tower class . . . . .	59



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all documented files with brief descriptions:

src/ <a href="#">bombProjectile.hpp</a>	65
src/ <a href="#">bombTower.hpp</a>	65
src/ <a href="#">bulletProjectile.hpp</a>	65
src/ <a href="#">bulletTower.hpp</a>	66
src/ <a href="#">button.hpp</a>	66
src/ <a href="#">enemy.hpp</a>	66
src/ <a href="#">explosion.hpp</a>	68
src/ <a href="#">freezingTower.hpp</a>	68
src/ <a href="#">game.hpp</a>	69
src/ <a href="#">levelManager.hpp</a>	70
src/ <a href="#">map.hpp</a>	71
src/ <a href="#">menu.hpp</a>	71
src/ <a href="#">missileProjectile.hpp</a>	72
src/ <a href="#">missileTower.hpp</a>	72
src/ <a href="#">path.hpp</a>	72
src/ <a href="#">player.hpp</a>	73
src/ <a href="#">poisonTower.hpp</a>	74
src/ <a href="#">projectile.hpp</a>	74
src/ <a href="#">resource_container.hpp</a>	75
src/ <a href="#">tower.hpp</a>	75





## Chapter 6

# Class Documentation

### 6.1 BombProjectile Class Reference

a projectile that causes damage to multiple enemies

```
#include <bombProjectile.hpp>
```

Inheritance diagram for BombProjectile:

### 6.2 BombTower Class Reference

Represents the [BombTower](#) class.

```
#include <bombTower.hpp>
```

Inheritance diagram for BombTower:

Collaboration diagram for BombTower:

#### Public Member Functions

- [BombTower](#) (sf::Vector2f)  
*Constructs a [BombTower](#) object at the specified position.*
- void [update](#) (std::list< std::shared\_ptr< [Enemy](#) > > &enemies, sf::Time time) override  
*Override of the base class method.*
- [BombProjectile](#) \* [shoot](#) () override  
*Override of the base class method to produce a [BombProjectile](#).*

#### 6.2.1 Detailed Description

Represents the [BombTower](#) class.

The [BombTower](#) is a specialized tower that shoots [BombProjectile](#) -projectiles. It is derived from the base [Tower](#) class and inherits common tower functionalities. Bomb tower can only lock enemies of ground type. BombProjectiles can, however, damage enemies of any type within explosion range of a bomb projectile (explosion range is dictated solely by bomb projectile objects).

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 BombTower()

```
BombTower::BombTower (
    sf::Vector2f position )
```

Constructs a [BombTower](#) object at the specified position.

#### Parameters

<i>position</i>	The initial position of the <a href="#">BombTower</a> (mouse position passed by the caller).
-----------------	--

Uses base [Tower](#) constructor.

## 6.2.3 Member Function Documentation

### 6.2.3.1 shoot()

```
BombProjectile * BombTower::shoot ( ) [override], [virtual]
```

Override of the base class method to produce a [BombProjectile](#).

#### Returns

[BombProjectile](#)\* A pointer to the created [BombProjectile](#) object.

[shoot \( \)](#) method calculates the direction towards locked enemy, normalizes it, and creates a [BombProjectile](#) that takes normalized direction, tower's position, damage, and locking range of the tower as arguments.

Implements [Tower](#).

### 6.2.3.2 update()

```
void BombTower::update (
    std::list< std::shared_ptr< Enemy > > & enemies,
    sf::Time time ) [override], [virtual]
```

Override of the base class method.

## Parameters

<i>enemies</i>	List of enemies passed from calling <code>Game::update</code> method.
<i>time</i>	Argument passed from calling <code>Game::update</code> method and is used to update <code>fireTimer_</code> .

This override for `update()` is very similar to `update()` method of base `Tower` class. The only difference is that it also checks `EnemyType` of an enemy as `BombTower` can only lock on enemies of `EnemyType::Ground`.

Reimplemented from `Tower`.

The documentation for this class was generated from the following files:

- `src/bombTower.hpp`
- `src/bombTower.cpp`

## 6.3 BulletProjectile Class Reference

a projectile that travels in a straight line and can hit only one enemy

```
#include <bulletProjectile.hpp>
```

Inheritance diagram for `BulletProjectile`:

Collaboration diagram for `BulletProjectile`:

### Public Member Functions

- **BulletProjectile** (`sf::Vector2f` shootDirection, `sf::Vector2f` position, `int` damage, `float` range)
- `bool` **hasHitEnemy** (`std::shared_ptr< Enemy >` &enemy) override  
*Checks if the bullet has hit an enemy. If the bullet's and enemy's sprites intersect, there has been a hit and the bullet causes damage to the enemy.*
- `void` **update** (`Game` &game) override
- `Textures::ProjectileID` **textureType** () override
- `float` **rotationAngle** () const  
*Calculates the rotation angle of the bullet based on its shooting direction !!! what is it used for.*

### 6.3.1 Detailed Description

a projectile that travels in a straight line and can hit only one enemy

### 6.3.2 Member Function Documentation

#### 6.3.2.1 hasHitEnemy()

```
bool BulletProjectile::hasHitEnemy (
    std::shared_ptr< Enemy > & enemy ) [override], [virtual]
```

Checks if the bullet has hit an enemy. If the bullet's and enemy's sprites intersect, there has been a hit and the bullet causes damage to the enemy.

#### Returns

true if bullet has hit an enemy.

**Parameters**

<i>enemy</i>	is a reference to an <a href="#">Enemy</a> object
--------------	---

Implements [Projectile](#).

**6.3.2.2 textureType()**

```
Textures::ProjectileID BulletProjectile::textureType ( ) [inline], [override], [virtual]
```

**Returns**

the texture ID of the type this derived class uses.

Implements [Projectile](#).

**6.3.2.3 update()**

```
void BulletProjectile::update (
    Game & game ) [override], [virtual]
```

If the bullet has gone out of range (exceeded its maximum distance), it's destroyed. Otherwise it goes through all enemies in the game to see if it has hit any one. If it has hit an enemy, the bullet is destroyed and the checking is stopped. If nothing of the before mentioned has happened, the bullet is moved.

**Parameters**

<i>game</i>	is a reference to the running game instance
-------------	---

Implements [Projectile](#).

The documentation for this class was generated from the following files:

- src/bulletProjectile.hpp
- src/bulletProjectile.cpp

**6.4 BulletTower Class Reference**

Represents the [BulletTower](#) class.

```
#include <bulletTower.hpp>
```

Inheritance diagram for BulletTower:

Collaboration diagram for BulletTower:

## Public Member Functions

- [BulletTower](#) (sf::Vector2f position)  
*Constructs a [BulletTower](#) object at the specified position.*
- [BulletProjectile](#) \* shoot () override  
*Override of the base class method to produce a [BulletProjectile](#).*

### 6.4.1 Detailed Description

Represents the [BulletTower](#) class.

The [BulletTower](#) is a specialized tower that shoots [BulletProjectile](#) -projectiles. It is derived from the base [Tower](#) class and inherits common tower functionalities.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 BulletTower()

```
BulletTower::BulletTower (  
    sf::Vector2f position )
```

Constructs a [BulletTower](#) object at the specified position.

##### Parameters

<i>position</i>	The initial position of the <a href="#">BulletTower</a> (mouse position passed by the caller).
-----------------	--

Uses base [Tower](#) constructor.

### 6.4.3 Member Function Documentation

#### 6.4.3.1 shoot()

```
BulletProjectile * BulletTower::shoot ( ) [override], [virtual]
```

Override of the base class method to produce a [BulletProjectile](#).

**Returns**

BulletProjectile\* A pointer to the created [BulletProjectile](#) object.

[shoot\(\)](#) method calculates the direction towards locked enemy, normalizes it, and creates a [BulletProjectile](#) that takes normalized direction, tower's position, damage, and slightly increased locking range of the tower as arguments.

Implements [Tower](#).

The documentation for this class was generated from the following files:

- src/bulletTower.hpp
- src/bulletTower.cpp

## 6.5 Button Class Reference

Represents a clickable button.

```
#include <button.hpp>
```

Inheritance diagram for Button:

Collaboration diagram for Button:

### Public Member Functions

- [Button](#) (Actions action, sf::Texture &texture, sf::Vector2f position, std::string text, sf::Font &font)  
*Constructs a button.*
- bool [isClicked](#) (sf::Vector2f mousePos) const  
*checks if the button has been clicked.*
- Actions [getAction](#) () const
- sf::Text [getLabel](#) () const

### Private Attributes

- Actions [action\\_](#)
- sf::Text [label\\_](#)

#### 6.5.1 Detailed Description

Represents a clickable button.

#### 6.5.2 Constructor & Destructor Documentation

##### 6.5.2.1 Button()

```
Button::Button (
    Actions action,
    sf::Texture & texture,
    sf::Vector2f position,
    std::string text,
    sf::Font & font ) [inline]
```

Constructs a button.

## Parameters

<i>action</i>	is the Actions enum determining the button type
<i>texture</i>	is the texture for the button
<i>position</i>	is the button position
<i>text</i>	is the button label text
<i>font</i>	is the font used for the button

### 6.5.3 Member Function Documentation

#### 6.5.3.1 isClicked()

```
bool Button::isClicked (
    sf::Vector2f mousePos ) const [inline]
```

checks if the button has been clicked.

## Returns

true if button was clicked, false otherwise.

The documentation for this class was generated from the following file:

- src/button.hpp

## 6.6 Enemy Class Reference

Inheritance diagram for Enemy:

Collaboration diagram for Enemy:

### Public Member Functions

- [Enemy](#) (int [hp](#), int [speed](#), EnemyType [type](#), int money, std::queue< sf::Vector2f > waypoints)
- void [update](#) (sf::Time time)
 

*Update function for enemies, updates enemy positions based on movement, and manages/applies status effects.*
- sf::Vector2f [getCenter](#) ()
- sf::Vector2f [getLocation](#) ()
- bool [dead](#) ()
- int [hp](#) ()
- int [initialHp](#) ()
- float [speed](#) ()
- int [poisonStatus](#) ()
- sf::Time [slowedStatus](#) ()
- EnemyType [type](#) ()

- void **takeDamage** (int damage)  
*//damages the enemy, takes in a damage value as a parameter, if the damage is higher than the health the enemy is automatically killed*
- void **kill** ()  
*kills the enemy, sets dead variable to true*
- void **applyPoison** (int stacksOfPoison, int damagePerStack)  
*applies poison status effect to enemies*
- void **applySlowed** (sf::Time duration, float slowCoefficient)  
*applies slowed status effect to enemies*
- void **slowedDamage** ()
- void **setVelocity** ()  
*sets the enemy velocity based on where the current waypoint is*
- bool **isWaypointPassed** (sf::Vector2f movement)  
*checks to see if the enemies current waypoint will be passed, this is determined by the movement variable of the enemy*
- void **findNewWaypoint** ()  
*finds a new waypoint for the enemy, this function goes through the waypoints queue and sets the current waypoint as the next waypoint in the queue if waypoints are empty it means the enemy has reached the castle and the enemy is set to state dead*
- std::queue< sf::Vector2f > **getWaypoints** ()
- void **moveEnemy** (sf::Vector2f movement)
- int **getMoney** () const
- void **updateHealthText** (const sf::Font &font)  
*updates the health text above enemies with the enemies current health*
- const sf::Text & **getHealthText** () const

## Private Attributes

- int **hp\_**
- int **initialHp\_**
- bool **dead\_** = false
- float **speed\_**
- float **actualSpeed\_**
- float **effectiveSpeed\_**
- sf::Text **healthText\_**
- EnemyType **type\_**
- int **poison\_** = 0
- sf::Time **slowed\_** = sf::Time::Zero
- int **money\_**
- sf::Vector2f **velocity\_**
- std::queue< sf::Vector2f > **waypoints\_**
- sf::Vector2f **currentWaypoint\_**
- int **direction\_**
- int **poisonDamage** = 0
- sf::Time **poisonTimer\_**
- float **slowCoefficient\_** = 0.f

## 6.6.1 Constructor & Destructor Documentation



### 6.6.1.1 Enemy()

```
Enemy::Enemy (
    int hp,
    int speed,
    EnemyType type,
    int money,
    std::queue< sf::Vector2f > waypoints ) [inline]
```

Initialises an enemy

#### Parameters

<i>hp</i>	reference to the health of the enemy
<i>speed</i>	reference to the speed of the enemy
<i>type</i>	reference to the enemy type
<i>money</i>	reference to the amount of money the enemy is worth
<i>waypoints</i>	reference to the waypoints for the enemy to take

## 6.6.2 Member Function Documentation

### 6.6.2.1 dead()

```
bool Enemy::dead ( )
```

#### Returns

returns boolean on the sate of the enemy, false if alive true if dead

### 6.6.2.2 getCenter()

```
sf::Vector2f Enemy::getCenter ( )
```

#### Returns

returns an sf::Vector2f corresponding to the enemies positional centre

#### 6.6.2.3 getHealthText()

```
const sf::Text & Enemy::getHealthText ( ) const
```

##### Returns

returns the healthText

#### 6.6.2.4 getLocation()

```
sf::Vector2f Enemy::getLocation ( )
```

##### Returns

returns the enemies location as a sf::Vector2f

#### 6.6.2.5 getMoney()

```
int Enemy::getMoney ( ) const
```

##### Returns

returns the amount of money this enemy provides when killed

#### 6.6.2.6 getWaypoints()

```
std::queue< sf::Vector2f > Enemy::getWaypoints ( )
```

##### Returns

returns waypoints

#### 6.6.2.7 hp()

```
int Enemy::hp ( )
```

##### Returns

returns enemy hp

### 6.6.2.8 initialHp()

```
int Enemy::initialHp ( )
```

#### Returns

returns enemies initialHP, this is used for the health text, as it displays the enemies health as a fraction over the initial health

### 6.6.2.9 isWaypointPassed()

```
bool Enemy::isWaypointPassed (
    sf::Vector2f movement )
```

checks to see if the enemies current waypoint will be passed, this is determined by the movement variable of the enemy

#### Returns

returns a bool

### 6.6.2.10 poisonStatus()

```
int Enemy::poisonStatus ( )
```

#### Returns

returns the duration of poison status effect

### 6.6.2.11 setVelocity()

```
void Enemy::setVelocity ( )
```

sets the enemy velocity based on where the current waypoint is

### 6.6.2.12 slowedStatus()

```
sf::Time Enemy::slowedStatus ( )
```

#### Returns

returns the duration of slowed status effect

#### 6.6.2.13 speed()

```
float Enemy::speed ( )
```

##### Returns

returns enemies speed

#### 6.6.2.14 type()

```
EnemyType Enemy::type ( )
```

##### Returns

returns enemy type

#### 6.6.2.15 update()

```
void Enemy::update (
    sf::Time time )
```

Update function for enemies, updates enemy positions based on movement, and manages/applies status effects.

The documentation for this class was generated from the following files:

- src/enemy.hpp
- src/enemy.cpp

## 6.7 Explosion Class Reference

Small class for drawing bomb explosions.

```
#include <explosion.hpp>
```

Inheritance diagram for Explosion:

Collaboration diagram for Explosion:

### Public Member Functions

- [Explosion](#) (int blastRange, sf::Vector2f pos)  
*Constructs an explosion.*
- void [update](#) (sf::Time inputtime)  
*Updates the explosion.*
- bool [isDone](#) ()  
*Return done\_ which tells if the explosion is done.*

## Private Attributes

- `sf::Time` **time\_**
- `int` **blastRange\_**
- `bool` **done\_**

### 6.7.1 Detailed Description

Small class for drawing bomb explosions.

See also

[BombProjectile](#)

### 6.7.2 Constructor & Destructor Documentation

#### 6.7.2.1 Explosion()

```
Explosion::Explosion (
    int blastRange,
    sf::Vector2f pos ) [inline]
```

Constructs an explosion.

Parameters

<i>blastRange</i>	Stores the bomb's blast range
<i>pos</i>	The bomb's position

### 6.7.3 Member Function Documentation

#### 6.7.3.1 update()

```
void Explosion::update (
    sf::Time inputtime ) [inline]
```

Updates the explosion.

Scales the circle and reduces time left. If the time (1 second) is over, sets the flag `done_`

## Parameters

<i>inputtime</i>	Time between frames from Game::getTime()
------------------	--

The documentation for this class was generated from the following file:

- src/explosion.hpp

## 6.8 FreezingTower Class Reference

Represents the Freezing [Tower](#) class.

```
#include <freezingTower.hpp>
```

Inheritance diagram for FreezingTower:

Collaboration diagram for FreezingTower:

### Public Member Functions

- [FreezingTower](#) (sf::Vector2f)  
*Constructs a [FreezingTower](#) object at the specified position.*
- void [update](#) (std::list< std::shared\_ptr< [Enemy](#) > > &enemies, sf::Time time) override  
*Override of the base class method.*
- [Projectile](#) \* [shoot](#) () override  
*Override of the base class method to that applies the slowing effect on enemies.*
- void [upgradeTower](#) () override  
*Override of the base class [upgradeTower\(\)](#) method.*

### Private Attributes

- std::list< std::shared\_ptr< [Enemy](#) > > [lockedEnemies\\_](#)  
*List of enemies currently locked by the [FreezingTower](#).*
- float [slowCoefficient\\_](#) = 0.2  
*Determines the strength of the slowing effect.*

### 6.8.1 Detailed Description

Represents the Freezing [Tower](#) class.

The Freezing [Tower](#) is a specialized non-damaging tower that slows down all the enemies within its range. The slowing effect is accomplished by applying it directly on enemies (rather than creating a projectile). Slowing effect affects all types of enemies.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 FreezingTower()

```
FreezingTower::FreezingTower (
    sf::Vector2f position )
```

Constructs a [FreezingTower](#) object at the specified position.

## Parameters

<i>position</i>	The initial position of the <a href="#">FreezingTower</a> (mouse position passed by the caller).
-----------------	--

Uses base [Tower](#) constructor.

### 6.8.3 Member Function Documentation

#### 6.8.3.1 shoot()

```
Projectile * FreezingTower::shoot ( ) [override], [virtual]
```

Override of the base class method to that applies the slowing effect on enemies.

## Returns

Projectile\* This override of [shoot\(\)](#) method always returns `nullptr`.

Applies the slowing effect on every enemy within `lockedEnemies_` container. As this method doesn't actually produce a projectile and the slowing effect is applied directly on the enemy, return value of this method is always `nullptr`.

Implements [Tower](#).

#### 6.8.3.2 update()

```
void FreezingTower::update (
    std::list< std::shared_ptr< Enemy > > & enemies,
    sf::Time time ) [override], [virtual]
```

Override of the base class method.

## Parameters

<i>enemies</i>	List of enemies passed from calling <a href="#">Game::update</a> method.
<i>time</i>	Argument passed from calling <a href="#">Game::update</a> method and is used to update <code>fireTimer_</code> .

This override for [update\(\)](#) first updates `fireTimer_`. Then `lockedEnemies_` container is cleared and `lockedEnemy_` is set to `nullptr`. After that `enemies` container is iterated through and enemies within tower's range are added to `lockedEnemies_` container. If at this point `lockedEnemies_` is not empty, `lockedEnemy_` is set with first pointer in `std::list<std::shared_ptr<Enemy>>` `lockedEnemies_`.

Reimplemented from [Tower](#).

### 6.8.3.3 upgradeTower()

```
void FreezingTower::upgradeTower ( ) [override], [virtual]
```

Override of the base class [upgradeTower\(\)](#) method.

Since [FreezingTower](#) is a non-damaging tower class its upgrade has to be overridden. As opposed to base [upgradeTower\(\)](#) method, upgrade of a [FreezingTower](#) increases `slowCoefficient_` rather than `damage_`.

Reimplemented from [Tower](#).

## 6.8.4 Member Data Documentation

### 6.8.4.1 lockedEnemies\_

```
std::list<std::shared_ptr<Enemy> > FreezingTower::lockedEnemies_ [private]
```

List of enemies currently locked by the [FreezingTower](#).

This list holds shared pointers to [Enemy](#) objects that the slowing effect will be applied to.

### 6.8.4.2 slowCoefficient\_

```
float FreezingTower::slowCoefficient_ = 0.2 [private]
```

Determines the strength of the slowing effect.

The `slowCoefficient_` represents the factor by which the movement speed of enemies is reduced when affected by the slowing effect.

The documentation for this class was generated from the following files:

- `src/freezingTower.hpp`
- `src/freezingTower.cpp`

## 6.9 Game Class Reference

This class runs the game logic.

```
#include <game.hpp>
```

Collaboration diagram for Game:



## Public Member Functions

- void `run` ()  
*this function is called from the main function to run the game.*
- `path` & `getPath` ()  
*Returns the path, which enemies follow.*

## Public Attributes

- `Map` `map`

## Private Member Functions

- void `processEvents` ()  
*processes user input*
- void `update` ()  
*Updates the state of objects in the game.*
- void `render` ()  
*Renders all objects onto the window.*
- void `loadTextures` ()  
*Helper function called in constructor, loads all textures.*
- void `createPath` ()  
*Used for testing the game, creates a hardcoded path.*
- void `checkTowers` ()  
*Check if a tower has been clicked.*
- void `testEnemy` ()
- void `testEnemySplit` (sf::Vector2f position, std::queue< sf::Vector2f > waypoints)
- void `updateMenus` ()  
*Helper function for updating the menus in game, called in `update()`.*
- sf::Time `getTime` () const

## Private Attributes

- sf::Clock `clock_`
- sf::Time `time_`
- sf::RenderWindow `window_`
- std::list< std::shared\_ptr< `Tower` > > `towers_`
- std::list< std::shared\_ptr< `Enemy` > > `enemies_`
- std::list< `Projectile` \* > `projectiles_`
- std::list< `Explosion` \* > `explosions_`
- `path` `path_`
- bool `dragged_`
- bool `paused_`  
*Indicates if a tower is currently being dragged into place.*
- bool `isGameOver_` = false  
*Is the game paused.*
- bool `isGameFinished_` = false  
*Is the game over because the player has died to an enemy.*
- sf::Font `font_`  
*Completed game.*

- sf::Text **gameOverText**  
*Stores text font.*
- sf::Text **gameFinishedText**
- sf::Sprite **castle\_sprite\_**
- std::unique\_ptr< [Menu](#) > **shop\_**
- std::unique\_ptr< [Menu](#) > **alternativeMenu\_**  
*Shop on left side.*
- std::shared\_ptr< [Tower](#) > **activeTower\_**  
*stores menu for upgrading, beginning game, and advancing to next level*
- bool **menuInactive** = false  
*Pointer to tower that is being upgraded or dragged into place.*
- [ResourceContainer](#)< Textures::TowerID, sf::Texture > **tower\_textures\_**  
*Indicates if the alternative menu is closed and needs to be deleted.*
- [ResourceContainer](#)< Textures::EnemyID, sf::Texture > **enemy\_textures\_**
- [ResourceContainer](#)< Textures::ProjectileID, sf::Texture > **projectile\_textures\_**
- [ResourceContainer](#)< Textures::Various, sf::Texture > **various\_textures\_**
- [Player](#) **player\_**
- [LevelManager](#) **levelManager\_**

## Friends

- class **Tower**
- class **BulletTower**
- class **BombTower**
- class **MissileTower**
- class **FreezingTower**
- class **BombProjectile**
- class **BulletProjectile**
- class **MissileProjectile**
- class **PoisonTower**
- class **Menu**
- class **LevelManager**

## 6.9.1 Detailed Description

This class runs the game logic.

## 6.9.2 Member Function Documentation

### 6.9.2.1 checkTowers()

```
void Game::checkTowers ( ) [private]
```

Check if a tower has been clicked.

If the mouse button has been pressed but no [Button](#) object was clicked, this checks if a purchased tower has been clicked. If a tower has been clicked, creates an upgrade menu, for upgrading or selling the tower.

### 6.9.2.2 getPath()

```
path & Game::getPath ( )
```

Returns the path, which enemies follow.

#### Returns

path& the path

### 6.9.2.3 processEvents()

```
void Game::processEvents ( ) [private]
```

processes user input

Gets widow events from SFML and checks if the window has been closed, or if the mouse button has been pressed. If the mouse button has been pressed checks if a button has been pressed by using [Menu::checkButtons\(\)](#) and checks if a tower has been clicked to open the upgrade menu.

#### See also

[checkTowers\(\)](#)

### 6.9.2.4 render()

```
void Game::render ( ) [private]
```

Renders all objects onto the window.

Clears window then draws objects. First draws the background and path, then iterates over towers, projectiles, enemies and explosions. Then draws some miscallenous things, like the tower being dragged if it exists and it's range. Menus are drawn last so they do not end up under anything.

### 6.9.2.5 run()

```
void Game::run ( )
```

this function is called from the main function to run the game.

If the window remains open, calls [processEvents\(\)](#), [update\(\)](#), and [render\(\)](#) in this order.

#### See also

[processEvents\(\)](#)

[update\(\)](#)

[render\(\)](#)

### 6.9.2.6 update()

```
void Game::update ( ) [private]
```

Updates the state of objects in the game.

First resets the timer, then handles updating objects by using their update functions.

### 6.9.2.7 updateMenus()

```
void Game::updateMenus ( ) [private]
```

Helper function for updating the menus in game, called in [update\(\)](#).

If a tower is being dragged calls [Menu::drag\(\)](#) to update it's position. Then updates the texts on screen. If an alternative menu has been closed deletes the alternative menu.

The documentation for this class was generated from the following files:

- [src/game.hpp](#)
- [src/game.cpp](#)

## 6.10 LevelManager Class Reference

Handles the creation and managing of levels.

```
#include <levelManager.hpp>
```

Collaboration diagram for LevelManager:

### Public Types

- using **variantData** = `std::variant< int, float, std::vector< int > >`  
*To allow the map holding level information to use different types.*

### Public Member Functions

- [LevelManager](#) (const std::string &src, [path](#) &path, [Game](#) &game, [Player](#) &player)
- int [getCurrentLevel](#) () const
- int [getLevelTotal](#) () const
- void [update](#) ()
- bool [readingSuccessful](#) ()

### Private Member Functions

- void [readLevels](#) ()
- void [initiateEnemies](#) ()

*Initiates the amount of enemies that is allowed for the level. Randomly chooses which type of enemy to initiate based on the allowed types for the level. Uses a switch case to initiate the right kind of enemy and adds it to the container of enemies. Resets the wait time and decreases waves.*

## Private Attributes

- `std::vector< std::map< std::string, variantData > > levelSpecs_`

*Container to hold all the levels. One entry in the outer container (vector) is one level, meaning index 0 is level one. The inner map holds all information regarding the specific level.*

*Keys:*

- `int currLevel_`
- `const std::string & src_`
- `bool readingSuccess_`
- `int levelTotal_`
- `float waitTime_`
- `path & path_`
- `Game & game_`
- `Player & player_`

### 6.10.1 Detailed Description

Handles the creation and managing of levels.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 LevelManager()

```
LevelManager::LevelManager (
    const std::string & src,
    path & path,
    Game & game,
    Player & player ) [inline]
```

Initialises a levelManager and reads the level information from file. Initial current level is zero (= level one) to follow indexing convention of level specifications container, to allow easier accessing

#### Parameters

<i>src</i>	is the source of the level information file that is to be read
<i>path</i>	is a reference to the path instance that creates the path of the game
<i>game</i>	is a reference to the running game instance
<i>player</i>	is a reference to the player instance of the game

See also

[readLevels\(\)](#)

### 6.10.3 Member Function Documentation

#### 6.10.3.1 `getCurrentLevel()`

```
int LevelManager::getCurrentLevel ( ) const
```

##### Returns

the current level

#### 6.10.3.2 `getLevelTotal()`

```
int LevelManager::getLevelTotal ( ) const
```

##### Returns

the total number of levels definend

#### 6.10.3.3 `readingSuccessfull()`

```
bool LevelManager::readingSuccessfull ( )
```

returns status flag for reading level info from file.

##### Returns

True if reading was successfull, false if not

#### 6.10.3.4 `readLevels()`

```
void LevelManager::readLevels ( ) [private]
```

Reads from the source file provided in constructor. Disregards first line of file as it is the formatting example. Then reads one line at a time:

- number of enemies per wave, number of waves, wait time between waves into variables
- allowed enemy types into a vector  
Adds the collected values into a map which gets pushed into the vector container that holds all levels.

### 6.10.3.5 update()

```
void LevelManager::update ( )
```

Updates the level manager, called while game is running. Counts down the wait time between waves of enemies. Initiates more enemies once wait time becomes zero, if there are waves left for the level. Moves to a new level once previous is complete and there are no enemies left.

See also

[initiateEnemies\(\)](#)

## 6.10.4 Member Data Documentation

### 6.10.4.1 levelSpecs\_

```
std::vector<std::map<std::string, variantData> > LevelManager::levelSpecs_ [private]
```

Container to hold all the levels. One entry in the outer container (vector) is one level, meaning index 0 is level one. The inner map holds all information regarding the specific level.

Keys:

.

- "enemyAmount" : the number of enemies allowed per wave (int)
- "waves" : the number of waves of enemies allowed per level (int)
- "waitTime" : the time (in seconds) between waves (float)
- "enemyTypes" : a vector containing the types of enemies allowed for the level

See also

[Enemy](#) class' type enum EnemyType

[variantData](#) The container that stores all levels information

The documentation for this class was generated from the following files:

- src/levelManager.hpp
- src/levelManager.cpp

## 6.11 Map Class Reference

Inheritance diagram for Map:

Collaboration diagram for Map:

## Public Member Functions

- void **loadMap** (const std::string &fileName)

## Public Attributes

- sf::Texture **texture**
- sf::Sprite **background**
- std::vector< sf::FloatRect > **unBuildable**

## Private Member Functions

- void **draw** (sf::RenderTarget &target, sf::RenderStates states) const override

The documentation for this class was generated from the following files:

- src/map.hpp
- src/map.cpp

## 6.12 Menu Class Reference

Class for storing a collection of buttons, a menu.

```
#include <menu.hpp>
```

## Public Member Functions

- void **draw** (sf::RenderWindow &window)  
*Draws all the objects in the menu.*
- void **checkButtons** (Game \*game)  
*Checks if a button in the menu has been pressed.*
- void **createMenu** (MenuType menu, Game \*game)  
*Creates the buttons and texts of a menu.*
- void **update** (Player &player)  
*Updates the status of the menu.*
- void **drag** (Game \*game)  
*Implements drag&drop placing of towers.*
- void **drawRange** (Game \*game)  
*Draws active tower range.*

## Private Member Functions

- void **newTower** (std::shared\_ptr< Tower > tower, Game \*game)  
*Adds a new tower to the game, called in checkButtons.*
- bool **canBePlaced** (Game \*game)  
*Checks if a tower can be placed in its current location.*



## Private Attributes

- `std::list< Button > buttons_`
- `std::vector< sf::Text > texts_`
- `sf::RectangleShape bg_`

### 6.12.1 Detailed Description

Class for storing a collection of buttons, a menu.

### 6.12.2 Member Function Documentation

#### 6.12.2.1 canBePlaced()

```
bool Menu::canBePlaced (  
    Game * game ) [private]
```

Checks if a tower can be placed in its current location.

##### Parameters

<i>game</i>	Pointer to the game object
-------------	----------------------------

##### Returns

true, if the tower can be placed

#### 6.12.2.2 checkButtons()

```
void Menu::checkButtons (  
    Game * game )
```

Checks if a button in the menu has been pressed.

Checks if the mouse has clicked a button. If a button has been clicked calls `getAction()` on the button and does the corresponding action

##### Parameters

<i>game</i>	Pointer to the game object
-------------	----------------------------

### 6.12.2.3 createMenu()

```
void Menu::createMenu (
    MenuType menu,
    Game * game )
```

Creates the buttons and texts of a menu.

#### Parameters

<i>menu</i>	Enumerator which tells the type of menu being created
<i>game</i>	Poiner to the game object

### 6.12.2.4 drag()

```
void Menu::drag (
    Game * game )
```

Implements drag&drop placing of towers.

If the mouse button is still pressed, moves the tower so it follows the mouse if the button is no longer pressed, checks if the player has enough money for the tower and if it can be placed, and if the conditions are met adds the tower to the game object

#### Parameters

<i>game</i>	pointer to the game object
-------------	----------------------------

#### See also

[canBePlaced\(\)](#)

### 6.12.2.5 draw()

```
void Menu::draw (
    sf::RenderWindow & window )
```

Draws all the objects in the menu.

#### Parameters

<i>window</i>	window onto which the objects get drawn
---------------	---

#### 6.12.2.6 drawRange()

```
void Menu::drawRange (
    Game * game )
```

Draws active tower range.

##### Parameters

<i>game</i>	pointer to the game object
-------------	----------------------------

#### 6.12.2.7 newTower()

```
void Menu::newTower (
    std::shared_ptr< Tower > tower,
    Game * game ) [private]
```

Adds a new tower to the game, called in checkButtons.

##### Parameters

<i>tower</i>	Pointer to new tower being built
<i>game</i>	Pointer to game

#### 6.12.2.8 update()

```
void Menu::update (
    Player & player )
```

Updates the status of the menu.

Updates the texts containing the money the player has and the health

##### Parameters

<i>player</i>	Reference to the player object
---------------	--------------------------------

The documentation for this class was generated from the following files:

- src/menu.hpp
- src/menu.cpp

## 6.13 MissileProjectile Class Reference

A projectile that targets (follows) a specific enemy.

```
#include <missileProjectile.hpp>
```

Inheritance diagram for MissileProjectile:

Collaboration diagram for MissileProjectile:

## Public Member Functions

- `MissileProjectile` (sf::Vector2f position, int damage, std::shared\_ptr< [Enemy](#) > targetEnemy)
- bool `hasHitEnemy` (std::shared\_ptr< [Enemy](#) > &enemy) override
- void `update` ([Game](#) &game) override
- Textures::ProjectileID `textureType` () override

## Private Attributes

- std::shared\_ptr< [Enemy](#) > `targetEnemy_`

### 6.13.1 Detailed Description

A projectile that targets (follows) a specific enemy.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 MissileProjectile()

```
MissileProjectile::MissileProjectile (
    sf::Vector2f position,
    int damage,
    std::shared_ptr< Enemy > targetEnemy ) [inline]
```

Missile does not need a pre-calculated directional vector, as its direction needs to be re-calculated everytime before it moves, hence the `shootDirection` is (0,0).

#### Parameters

<i>targetEnemy</i>	is the enemy that the missile is targeting (following).
--------------------	---

### 6.13.3 Member Function Documentation

### 6.13.3.1 hasHitEnemy()

```
bool MissileProjectile::hasHitEnemy (
    std::shared_ptr< Enemy > & enemy ) [override], [virtual]
```

Checks whether the missile has hit its target or not. If the missile's and enemy's sprites intersect, there has been a hit and the missile causes damage to the enemy.

#### Returns

True if missile has hit it's target, otherwise false.

#### Parameters

<i>enemy</i>	is a reference to an <a href="#">Enemy</a> object, the missiles target.
--------------	---

Implements [Projectile](#).

### 6.13.3.2 textureType()

```
Textures::ProjectileID MissileProjectile::textureType ( ) [inline], [override], [virtual]
```

#### Returns

the texture ID of the type this derived class uses.

Implements [Projectile](#).

### 6.13.3.3 update()

```
void MissileProjectile::update (
    Game & game ) [override], [virtual]
```

Firstly makes sure that the target enemy still exists, if it doesn't the missile is destroyed. If the enemy still exists it checks whether or not the missile has hit it, if there's been a hit, the missile is destroyed. If the missile has not hit the enemy, it re-calculates its directional vector, based on its and the target enemy's current positions, and moves towards the target.

#### Parameters

<i>game</i>	is a reference to the running game instance.
-------------	--

Implements [Projectile](#).

The documentation for this class was generated from the following files:

- `src/missileProjectile.hpp`
- `src/missileProjectile.cpp`

## 6.14 MissileTower Class Reference

Represents the [MissileTower](#) class.

```
#include <missileTower.hpp>
```

Inheritance diagram for MissileTower:

Collaboration diagram for MissileTower:

### Public Member Functions

- [MissileTower](#) (`sf::Vector2f`)  
*Constructs a [MissileTower](#) object at the specified position.*
- [MissileProjectile](#) \* `shoot` () override  
*Override of the base class method to produce a [MissileProjectile](#).*

### 6.14.1 Detailed Description

Represents the [MissileTower](#) class.

The [MissileTower](#) is a specialized tower that shoots [MissileProjectile](#) -projectiles. It is derived from the base [Tower](#) class and inherits common tower functionalities.

### 6.14.2 Constructor & Destructor Documentation

#### 6.14.2.1 MissileTower()

```
MissileTower::MissileTower (
    sf::Vector2f position )
```

Constructs a [MissileTower](#) object at the specified position.

#### Parameters

<i>position</i>	The initial position of the <a href="#">MissileTower</a> (mouse position passed by the caller).
-----------------	---

Uses base [Tower](#) constructor.

### 6.14.3 Member Function Documentation

#### 6.14.3.1 shoot()

```
MissileProjectile * MissileTower::shoot ( ) [override], [virtual]
```

Override of the base class method to produce a [MissileProjectile](#).

##### Returns

MissileProjectile\* A pointer to the created [MissileProjectile](#) object.

[shoot\(\)](#) method creates a [MissileProjectile](#) that takes tower's position, damage and lockedEnemy as arguments.

Implements [Tower](#).

The documentation for this class was generated from the following files:

- src/missileTower.hpp
- src/missileTower.cpp

## 6.15 path Class Reference

### Public Member Functions

- [path](#) (const std::string &src)
- void [readPath](#) ()
- bool [readingSuccessful](#) ()
- void [addWaypoint](#) (const sf::Vector2f &point)
- std::queue< sf::Vector2f > [getWaypoints](#) () const
- void [makeUnBuildablePath](#) ()

### Public Attributes

- std::queue< sf::Vector2f > [waypoints\\_](#)
- std::vector< sf::Vector2f > [wayPoints](#)
- std::vector< sf::FloatRect > [unBuildable](#)
- std::vector< std::vector< sf::Vector2f > > [paths\\_](#)

*The container that stores all paths coordinates.*

### Static Public Attributes

- static const float [width](#) = 60.f

## Private Attributes

- `const std::string & src_`
- `bool readingSuccess_`

## Friends

- class `enemy`

## 6.15.1 Constructor & Destructor Documentation

### 6.15.1.1 `path()`

```
path::path (
    const std::string & src ) [inline]
```

Constructs a path by reading coordinate values from a file, randomly chooses one of the paths and adds the coordinates to the `waypoints` containers.

#### Parameters

<code>src</code>	is the source of the path information file to be read
------------------	---

#### See also

[readPath\(\)](#)

## 6.15.2 Member Function Documentation

### 6.15.2.1 `addWaypoint()`

```
void path::addWaypoint (
    const sf::Vector2f & point )
```

populates the waypoint que with all the waypoints required for the enemy class to traverse the path

### 6.15.2.2 `readingSuccessfull()`

```
bool path::readingSuccessfull ( )
```

returns status flag for reading level info from file.

#### Returns

True if reading was successfull, false if not



### 6.15.2.3 readPath()

```
void path::readPath ( )
```

Reads the source file provided in the constructor. Disregards the first line as it is the formatting example. Reads the values into a vector of SFML vector coordinates, and then adds that vector containing the path into a vector that contains all the paths from the file.

The documentation for this class was generated from the following files:

- src/path.hpp
- src/path.cpp

## 6.16 Player Class Reference

Class representing the player.

```
#include <player.hpp>
```

Inheritance diagram for Player:

Collaboration diagram for Player:

### Public Member Functions

- [Player](#) ()  
*Initialises a player with default values.*
- int [getWallet](#) () const
- int [getHP](#) () const
- int [getLevel](#) () const
- void [levelUp](#) ()  
*increases the players level by one*
- void [addMoney](#) (int amount)  
*adds money to the players wallet*
- void [removeMoney](#) (int cost)  
*removes money from the players wallet*
- void [removeHP](#) (int amount)  
*removes health points from the player*

### Private Attributes

- int [hp\\_](#)
- int [wallet\\_](#)
- int [level\\_](#)

### 6.16.1 Detailed Description

Class representing the player.

The class handles player health and money and stores the current level number.

## 6.16.2 Constructor & Destructor Documentation

### 6.16.2.1 Player()

```
Player::Player ( ) [inline]
```

Initialises a player with default values.

#### Parameters

<i>hp_</i>	is the health points of the player
<i>wallet</i> ↔ —	is how much money the player has
<i>level</i> ↔ —	is the level of the player

## 6.16.3 Member Function Documentation

### 6.16.3.1 addMoney()

```
void Player::addMoney (
    int amount )
```

adds money to the players wallet

#### Parameters

<i>amount</i>	is how much money is to be added
---------------	----------------------------------

### 6.16.3.2 getHP()

```
int Player::getHP ( ) const
```

#### Returns

how many health points the player has

### 6.16.3.3 getLevel()

```
int Player::getLevel ( ) const
```

#### Returns

the current level of the player

### 6.16.3.4 getWallet()

```
int Player::getWallet ( ) const
```

#### Returns

how much money the player has

### 6.16.3.5 removeHP()

```
void Player::removeHP (
    int amount )
```

removes health points from the player

#### Parameters

<i>amount</i>	is how much hp is to be removed
---------------	---------------------------------

### 6.16.3.6 removeMoney()

```
void Player::removeMoney (
    int cost )
```

removes money from the players wallet

#### Parameters

<i>cost</i>	is how much money is to be removed
-------------	------------------------------------

The documentation for this class was generated from the following files:

- src/player.hpp
- src/player.cpp

## 6.17 PoisonTower Class Reference

Represents the Poison [Tower](#) class.

```
#include <poisonTower.hpp>
```

Inheritance diagram for PoisonTower:

Collaboration diagram for PoisonTower:

### Public Member Functions

- [PoisonTower](#) (sf::Vector2f)  
*Constructs a [PoisonTower](#) object at the specified position.*
- void [update](#) (std::list< std::shared\_ptr< [Enemy](#) > > &enemies, sf::Time time) override  
*Override of the base class method.*
- [Projectile](#) \* [shoot](#) () override  
*Override of the base class method to that applies the poison effect on enemies.*

### Private Attributes

- std::list< std::shared\_ptr< [Enemy](#) > > [lockedEnemies\\_](#)  
*List of enemies currently locked by the [PoisonTower](#).*

#### 6.17.1 Detailed Description

Represents the Poison [Tower](#) class.

The [PoisonTower](#) is a specialized tower that applies the poison effect on all the enemies within its range. The poison effect is accomplished by applying it directly on enemies (rather than creating a projectile) and it deals damage over time (x damage every y units of time for the duration of z seconds). The poison effect affects all types of enemies.

#### 6.17.2 Constructor & Destructor Documentation

##### 6.17.2.1 PoisonTower()

```
PoisonTower::PoisonTower (
    sf::Vector2f position )
```

Constructs a [PoisonTower](#) object at the specified position.

Parameters

<i>position</i>	The initial position of the <a href="#">PoisonTower</a> (mouse position passed by the caller).
-----------------	--

Uses base [Tower](#) constructor.

### 6.17.3 Member Function Documentation

#### 6.17.3.1 shoot()

```
Projectile * PoisonTower::shoot ( ) [override], [virtual]
```

Override of the base class method to that applies the poison effect on enemies.

##### Returns

Projectile\* This override of [shoot\(\)](#) method always returns `nullptr`.

Applies the poison effect on every enemy within `lockedEnemies_` container. As this method doesn't actually produce a projectile and the poison effect is applied directly on the enemy, return value of this method is always `nullptr`.

Implements [Tower](#).

#### 6.17.3.2 update()

```
void PoisonTower::update (
    std::list< std::shared_ptr< Enemy > > & enemies,
    sf::Time time ) [override], [virtual]
```

Override of the base class method.

##### Parameters

<i>enemies</i>	List of enemies passed from calling <a href="#">Game::update</a> method.
<i>time</i>	Argument passed from calling <a href="#">Game::update</a> method and is used to update <code>fireTimer_</code> .

This override for [update\(\)](#) first updates `fireTimer_`. Then `lockedEnemies_` container is cleared and `lockedEnemy_` is set to `nullptr`. After that enemies container is iterated through and enemies within tower's range are added to `lockedEnemies_` container. If at this point `lockedEnemies_` is not empty, `lockedEnemy_` is set with first pointer in `std::list<std::shared_ptr<Enemy>>` `lockedEnemies_`.

Reimplemented from [Tower](#).

### 6.17.4 Member Data Documentation

### 6.17.4.1 lockedEnemies\_

```
std::list<std::shared_ptr<Enemy> > PoisonTower::lockedEnemies_ [private]
```

List of enemies currently locked by the [PoisonTower](#).

This list holds shared pointers to [Enemy](#) objects that the poison effect will be applied to.

The documentation for this class was generated from the following files:

- src/poisonTower.hpp
- src/poisonTower.cpp

## 6.18 Projectile Class Reference

```
#include <projectile.hpp>
```

Inheritance diagram for Projectile:

Collaboration diagram for Projectile:

### Public Member Functions

- [Projectile](#) (sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed, std::string type, float maxDistance)  
*Constructs a projectile and sets it's initial position.*
- virtual [~Projectile](#) ()  
*Destroy the [Projectile](#) object.*
- float [getSpeed](#) () const
- const std::string & [getType](#) () const
- int [getDamage](#) () const
- sf::Vector2f [getShootDir](#) () const
- void [destroy](#) ()
- bool [isDestroyed](#) ()  
*Returns wheter the projectile is destroyed, and needs to be deleted, or not.*
- bool [distToTower](#) ()  
*Calculates the projectiles distance from the tower that created it.*
- virtual bool [hasHitEnemy](#) (std::shared\_ptr< [Enemy](#) > &)=0  
*checks if the projectile has hit an enemy. Overriden in each derived class.*
- virtual void [update](#) ([Game](#) &)=0  
*updates the projectiles state as is defiened in each derived class*
- virtual Textures::ProjectileID [textureType](#) ()=0

### Private Attributes

- float [speed\\_](#)
- std::string [type\\_](#)
- int [damage\\_](#)
- sf::Vector2f [position\\_](#)
- float [maxDistance\\_](#)
- sf::Vector2f [shootDirection\\_](#)
- bool [isDestroyed\\_](#)

## 6.18.1 Detailed Description

An abstract class for deriving projectile like, "flying", objects.

## 6.18.2 Constructor & Destructor Documentation

### 6.18.2.1 Projectile()

```
Projectile::Projectile (
    sf::Vector2f shootDirection,
    sf::Vector2f position,
    int damage,
    float speed,
    std::string type,
    float maxDistance ) [inline]
```

Constructs a projectile and sets it's initial position.

#### Parameters

<i>shootDirection</i>	is the normalised directional vector used to move the projectile, determined by the creating tower
<i>position</i>	is position of the tower that created the projectile, is used as a starting position
<i>damage</i>	is the amount of damage that the projectile will cause the enemy it hits, determined by the creating tower
<i>speed</i>	is the speed at which the projectile moves, pre-defined for each derived type
<i>type</i>	is the type of the projectile, pre-defined for each derived type
<i>maxDistance</i>	is the maximum distance the projectile is allowed to move from it's tower, based on the towers range

## 6.18.3 Member Function Documentation

### 6.18.3.1 destroy()

```
void Projectile::destroy ( )
```

Sets the `isDestroyed_` flag to true when the projectile has hit an enemy, and fulfilled its purpose, or when it has gone out of range (exceeded its max distance), and needs to be destroyed.

### 6.18.3.2 distToTower()

```
bool Projectile::distToTower ( )
```

Calculates the projectiles distance from the tower that created it.

#### Returns

true if the projectile is at, or has exceeded, its maximum distance. False otherwise

### 6.18.3.3 getDamage()

```
int Projectile::getDamage ( ) const
```

#### Returns

the damage of the projectile

### 6.18.3.4 getShootDir()

```
sf::Vector2f Projectile::getShootDir ( ) const
```

#### Returns

the directional vector of the projectile

### 6.18.3.5 getSpeed()

```
float Projectile::getSpeed ( ) const
```

#### Returns

the speed of the projectile

### 6.18.3.6 getType()

```
const std::string & Projectile::getType ( ) const
```

#### Returns

the type of the projectile



### 6.18.3.7 hasHitEnemy()

```
virtual bool Projectile::hasHitEnemy (
    std::shared_ptr< Enemy > & ) [pure virtual]
```

checks if the projectile has hit an enemy. Overridden in each derived class.

Implemented in [BombProjectile](#), [BulletProjectile](#), and [MissileProjectile](#).

### 6.18.3.8 textureType()

```
virtual Textures::ProjectileID Projectile::textureType ( ) [pure virtual]
```

#### Returns

the ID of the texture the projectile type uses The return value is directly hardcoded in derived classes.

Implemented in [BombProjectile](#), [BulletProjectile](#), and [MissileProjectile](#).

### 6.18.3.9 update()

```
virtual void Projectile::update (
    Game & ) [pure virtual]
```

updates the projectiles state as is defined in each derived class

Implemented in [BombProjectile](#), [BulletProjectile](#), and [MissileProjectile](#).

The documentation for this class was generated from the following files:

- [src/projectile.hpp](#)
- [src/projectile.cpp](#)

## 6.19 ResourceContainer< T\_enum, T\_resource > Class Template Reference

Template container for textures etc resources.

```
#include <resource_container.hpp>
```

### Public Member Functions

- void [load](#) (T\_enum type, std::string filename)  
*Loads and stores a resource.*
- T\_resource & [get](#) (T\_enum type) const  
*Find and return requested resource.*

## Private Attributes

- `std::map< T_enum, std::unique_ptr< T_resource > > resources_`

### 6.19.1 Detailed Description

```
template<typename T_enum, typename T_resource>
class ResourceContainer< T_enum, T_resource >
```

Template container for textures etc resources.

### 6.19.2 Member Function Documentation

#### 6.19.2.1 get()

```
template<typename T_enum , typename T_resource >
T_resource & ResourceContainer< T_enum, T_resource >::get (
    T_enum type ) const [inline]
```

Find and return requested resource.

##### Parameters

<i>type</i>	Enumerator defining which texture is wanted
-------------	---

##### Returns

Returns reference to resource if found

#### 6.19.2.2 load()

```
template<typename T_enum , typename T_resource >
void ResourceContainer< T_enum, T_resource >::load (
    T_enum type,
    std::string filename ) [inline]
```

Loads and stores a resource.

##### Parameters

<i>type</i>	Enumerator which defines the type of this resource.
<i>filename</i>	path to file containing the resource.

The documentation for this class was generated from the following file:

- `src/resource_container.hpp`

## 6.20 Tower Class Reference

Represents abstract tower class.

```
#include <tower.hpp>
```

Inheritance diagram for Tower:

Collaboration diagram for Tower:

### Public Member Functions

- [Tower](#) (sf::Vector2f position, const std::string &type, int baseCost, float range, sf::Time fireRate, int damage, int upgradeCost)  
*Constructor for abstract tower is used in constructor for derived tower classes.*
- const std::string & **getType** () const
- const int **getBaseCost** () const
- sf::Time **getFireRate** () const
- const float **getRange** () const
- int **getDamage** () const
- std::shared\_ptr< [Enemy](#) > **getLockedEnemy** () const
- bool **isMaxLevelReached** () const
- int **getCurrentLvl** () const
- const int **getUpgradeCost** () const
- sf::Time **getFireTimer** ()
- bool **enemyWithinRange** (std::shared\_ptr< [Enemy](#) > enemy)  
*Check if the enemy is within the range of the tower.*
- virtual [Projectile](#) \* **shoot** ()=0  
*[shoot](#) () method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with [PoisonTower](#) and [FreezingTower](#)).*
- virtual void **upgradeTower** ()  
*[upgradeTower](#) () method is virtual as upgrade logic is same for all types of towers except [FreezingTower](#)*
- virtual void **update** (std::list< std::shared\_ptr< [Enemy](#) > > &enemies, sf::Time time)  
*[update](#) () method is virtual as some types of towers use method of the base class.*
- void **updateFireTimer** (sf::Time &dt)  
*Increments fireTimer\_ by dt.*
- void **setLevel** (int level)
- void **setMaxLevelFlag** ()
- void **setLockedEnemy** (std::shared\_ptr< [Enemy](#) > enemy)
- void **resetFireTimer** ()

## Private Attributes

- const std::string `type_`
- const int `baseCost_`
- const float `range_`
- int `damage_`
- int `currentLvl_`
- const int `upgradeCost_`
- std::shared\_ptr< `Enemy` > `lockedEnemy_`
- sf::Time `fireTimer_`
- sf::Time `fireRate_`
- bool `maxLevelReached_`

### 6.20.1 Detailed Description

Represents abstract tower class.

The `Tower` class is a base class for various types of towers, each with its unique characteristics. Towers can lock onto enemies within a specified range, shoot projectiles, and be upgraded to increase their effectiveness. This class acts as a common interface for managing towers and a foundation for derived tower classes.

### 6.20.2 Constructor & Destructor Documentation

#### 6.20.2.1 Tower()

```
Tower::Tower (
    sf::Vector2f position,
    const std::string & type,
    int baseCost,
    float range,
    sf::Time fireRate,
    int damage,
    int upgradeCost )
```

Constructor for abstract tower is used in constructor for derived tower classes.

#### Parameters

<i>position</i>	Determined by constructor of derived tower class
<i>type</i>	Determined by constructor of derived tower class
<i>baseCost</i>	Determined by constructor of derived tower class
<i>range</i>	Determined by constructor of derived tower class
<i>fireRate</i>	Determined by constructor of derived tower class
<i>damage</i>	Determined by constructor of derived tower class
<i>upgradeCost</i>	Determined by constructor of derived tower class

## 6.20.3 Member Function Documentation

### 6.20.3.1 enemyWithinRange()

```
bool Tower::enemyWithinRange (
    std::shared_ptr< Enemy > enemy )
```

Check if the enemy is within the range of the tower.

#### Parameters

<i>enemy</i>	A shared pointer to an <a href="#">Enemy</a> object passed from calling <a href="#">Tower::update</a> method.
--------------	---

#### Returns

`true` if locking range of the tower is more or equal to distance between the enemy and the tower.  
`false` otherwise.

### 6.20.3.2 shoot()

```
virtual Projectile * Tower::shoot ( ) [pure virtual]
```

[shoot \( \)](#) method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with [PoisonTower](#) and [FreezingTower](#)).

#### Returns

Projectile\*

Implemented in [BombTower](#), [BulletTower](#), [FreezingTower](#), [MissileTower](#), and [PoisonTower](#).

### 6.20.3.3 update()

```
void Tower::update (
    std::list< std::shared_ptr< Enemy > > & enemies,
    sf::Time time ) [virtual]
```

[update \( \)](#) method is virtual as some types of towers use method of the base class.

Main logic of tower.

First, we check whether currently locked enemy is not `nullptr`, not dead and still within tower's range. If this condition is satisfied nothing else is done. Otherwise, locked enemy is set to `nullptr` and `enemies` container is iterated through to find the fastest enemy which is within tower's range and alive. If there is no enemies alive within tower's range, `lockedEnemy_` member stays `nullptr`. Otherwise, `lockedEnemy_` is set to the pointer to the fastest, alive enemy within tower's range.

**Parameters**

<i>enemies</i>	List argument passed from calling <a href="#">Game::update</a> method.
<i>time</i>	Argument passed from calling <a href="#">Game::update</a> method and is used to update <a href="#">Tower::fireTimer_</a> .

Reimplemented in [BombTower](#), [FreezingTower](#), and [PoisonTower](#).

**6.20.3.4 updateFireTimer()**

```
void Tower::updateFireTimer (
    sf::Time & dt )
```

Increments `fireTimer_` by `dt`.

**Parameters**

<i>dt</i>	Time since last frame and is passed from <a href="#">Game::update()</a> .
-----------	---

**6.20.3.5 upgradeTower()**

```
void Tower::upgradeTower ( ) [virtual]
```

[upgradeTower\(\)](#) method is virtual as upgrade logic is same for all types of towers except [FreezingTower](#)

[FreezingTower](#)

This method upgrades tower by one level, increases its `damage_` member by 1.5 times and sets the maximum level flag to true.

**Note**

If the maximum level has already been reached, this method has no effect

Reimplemented in [FreezingTower](#).

**6.20.4 Member Data Documentation****6.20.4.1 type\_**

```
const std::string Tower::type_ [private]
```

## Parameters

<i>type_</i>	A string representing type of the tower
<i>baseCost_</i>	The base cost for the type of tower
<i>range_</i>	The enemy locking range of the tower
<i>damage_</i>	Damage of the tower that is passed as a parameter to projectile constructor
<i>currentLvl_</i>	Current level of the tower, initially set 1 and can be upgraded up to level 2
<i>upgradeCost_</i>	Set at $1.5 * \text{base cost of tower}$ for all types of towers.
<i>lockedEnemy_</i>	The locked enemy of the tower; initially set to nullptr.
<i>fireTimer_</i>	Member used to count how much time has passed since last shot.
<i>fireRate_</i>	Member that dictates how often tower can shoot projectiles (or apply other effect on enemies).
<i>maxLevelReached_</i>	Flag used to check whether tower is already at max level.

The documentation for this class was generated from the following files:

- src/tower.hpp
- src/tower.cpp





## Chapter 7

# File Documentation

### 7.1 bombProjectile.hpp

```
1 #ifndef BOMB_PROJECTILE
2 #define BOMB_PROJECTILE
3
4 #include "projectile.hpp"
5 #include <list>
6
10 class BombProjectile : public Projectile
11 {
12 private:
13     int blastRange_;
14 public:
15
19     BombProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range) // <- tbd
20     : Projectile(shootDirection, position, damage, 60.0, "bomb", range), blastRange_(1000) {}
21
28     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) override;
29
36     void update(Game& game) override;
37
41     Textures::ProjectileID textureType() override { return Textures::Bomb; }
42 };
43
44
45 #endif
```

### 7.2 bombTower.hpp

```
1 #ifndef BOMB_TOWER_H
2 #define BOMB_TOWER_H
3 #include "tower.hpp"
4 #include "bombProjectile.hpp"
15 class BombTower : public Tower {
16 public:
22     BombTower(sf::Vector2f);
29     void update(std::list<std::shared_ptr<Enemy>> &enemies, sf::Time time) override;
35     BombProjectile* shoot() override;
36 };
37 #endif //BOMB_TOWER
```

### 7.3 bulletProjectile.hpp

```
1 #ifndef BULLET_PROJECTILE
2 #define BULLET_PROJECTILE
3
4 #include "projectile.hpp"
5
9 class BulletProjectile : public Projectile
10 {
11 public:
```

```

12     BulletProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range)
13     : Projectile(shootDirection, position, damage, 500, "bullet", range) {}
14
22     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) override;
23
31     void update(Game& game) override;
32
36     Textures::ProjectileID textureType() override { return Textures::Bullet; }
37
42     float rotationAngle() const; //this one is used to calculate rotation angle of a projectile.
43 };
44
45
46 #endif

```

## 7.4 bulletTower.hpp

```

1 #ifndef BULLET_TOWER_H
2 #define BULLET_TOWER_H
3 #include "tower.hpp"
4 #include "bulletProjectile.hpp"
12 class BulletTower : public Tower {
13 public:
19     BulletTower(sf::Vector2f position);
25     BulletProjectile* shoot() override;
26 };
27 #endif //BULLET_TOWER

```

## 7.5 button.hpp

```

1 #ifndef BUTTON
2 #define BUTTON
3 #include <SFML/Graphics.hpp>
4
9 enum class Actions{
10     Tower1,
11     Tower2,
12     Tower3,
13     Tower4,
14     Tower5,
15     Pause,
16     Upgrade,
17     Sell,
18     Close, // In upgrade menu, closes upgrade menu.
19     Level // Click to start level
20 };
21
26 class Button : public sf::Sprite {
27 public:
37     Button(Actions action, sf::Texture& texture, sf::Vector2f position, std::string text, sf::Font& font)
38     : action_(action) {
39         setTexture(texture);
40         setPosition(position);
41         label_ = sf::Text(text, font, 15);
42         label_.setPosition(position.x, position.y+20);
43     }
49     bool isClicked(sf::Vector2f mousePos) const {
50         return getGlobalBounds().contains(mousePos);
51     }
52
53     Actions getAction() const { return action_; }
54     sf::Text getLabel() const { return label_; }
55
56 private:
57     Actions action_;
58     sf::Text label_;
59
60 };
61
62
63 #endif

```

## 7.6 enemy.hpp

```

1 #ifndef ENEMY_HPP

```

```

2 #define ENEMY_HPP
3 #include <string>
4 #include "path.hpp"
5 #include <queue>
6 #include "player.hpp"
7 #include <SFML/System/Vector2.hpp>
8 #include <SFML/Graphics.hpp>
9 #include <random>
10
11 enum class EnemyType {
12     Ground,
13     Flying,
14     Split,
15 };
16
17 class Enemy :public sf::Sprite {
18 public:
19
20     Enemy(int hp, int speed, EnemyType type, int money, std::queue<sf::Vector2f> waypoints)
21         : hp_(hp), actualSpeed_(speed), speed_(speed), effectiveSpeed_(speed), type_(type),
22         money_(money), waypoints_(waypoints), initialHp_(hp) {
23
24         // Random y value of starting pos, gets set as a negative value
25         // So enemies spawn outside window and then move in
26         //int rand_y = std::rand() % 40;
27
28         //tries to avoid enemies being on top of eachother
29         std::random_device rd;
30         std::uniform_int_distribution range(1,40);
31         int x = range(rd);
32         int y = range(rd);
33         setPosition(waypoints_.front() - sf::Vector2f(x,y));
34
35         if (!waypoints_.empty()) {
36             currentWaypoint_ = waypoints_.front();
37         }
38         setVelocity();
39     }
40
41     ~Enemy() {}
42     void update(sf::Time time);
43     sf::Vector2f getCenter();
44     sf::Vector2f getLocation();
45     bool dead();
46     int hp();
47     int initialHp();
48     float speed();
49     int poisonStatus();
50     sf::Time slowedStatus();
51     EnemyType type();
52     void takeDamage(int damage); //decreases the hp_ variable and if hp reaches 0 than the enemy is
53     automatically destroyed
54     void kill();
55     void applyPoison(int stacksOfPoison, int damagePerStack);
56
57     void applySlowed(sf::Time duration, float slowCoefficient);
58
59     void slowedDamage();
60     void setVelocity();
61     bool isWaypointPassed(sf::Vector2f movement);
62     void findNewWaypoint();
63     std::queue<sf::Vector2f> getWaypoints();
64
65     void moveEnemy(sf::Vector2f movement);
66     int getMoney() const;
67     void updateHealthText(const sf::Font& font);
68     const sf::Text& getHealthText() const;
69 private:
70     int hp_;
71
72     int initialHp_;
73
74     bool dead_ = false;
75
76     float speed_;
77
78     float actualSpeed_;
79
80     float effectiveSpeed_;
81
82     sf::Text healthText_;
83
84     EnemyType type_;
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165

```

```

166     int poison_ = 0; //If poison is larger than 0 that means that the enemy is poisoned
167     // the length of time that the enemy is poisoned for depends on how large the poison
168     //value is as the number decreases incremently until 0
169     sf::Time slowed_ = sf::Time::Zero;
170     //How much money the player recieves for killing the monster
171     int money_;
172     //waypoint based movement, the path class provides a queue of waypoints that take the enemies
    through the path to the end
173
174     sf::Vector2f velocity_;
175
176     std::queue<sf::Vector2f> waypoints_;
177
178     sf::Vector2f currentWaypoint_;
179
180     int direction_; //0 = down, 1= left, 2= right, 3 = up
181
182     int poisonDamage = 0;
183
184     sf::Time poisonTimer_;
185
186     float slowCoefficient_ = 0.f;
187 };
188
189 #endif

```

## 7.7 explosion.hpp

```

1 #ifndef EXPLOSION
2 #define EXPLOSION
3 #include <SFML/Graphics.hpp>
4 #include <SFML/System.hpp>
5 #include <stdio.h>
6 #define BOMB_SIZE_HALF 24
7
14 class Explosion : public sf::CircleShape {
15 public:
22     Explosion(int blastRange, sf::Vector2f pos) : blastRange_(blastRange), done_(false) {
23         time_ = sf::seconds(1);
24         setPosition(pos.x + BOMB_SIZE_HALF, pos.y + BOMB_SIZE_HALF);
25         setRadius(2);
26         setOrigin(2, 2);
27         setFillColor(sf::Color(255, 64, 0, 150));
28     }
29
38     void update(sf::Time inputtime) {
39         time_ -= inputtime;
40         if (time_ < sf::microseconds(0)) {
41             done_ = true;
42             std::cout << "The explosion is done" << std::endl;
43             return;
44         }
45         if (time_ >= sf::seconds(0.5)) {
46             setScale(getScale().x + 1, getScale().y + 1);
47         } else {
48             setScale(getScale().x - 1, getScale().y - 1);
49         }
50     }
51
52     bool isDone(){ return done_; }
53
54 private:
55     sf::Time time_;
56     int blastRange_;
57     bool done_;
58
59 };
60
61 #endif

```

## 7.8 freezingTower.hpp

```

1 #ifndef FREEZING_TOWER
2 #define FREEZING_TOWER
3 #include "tower.hpp"
4 #include "enemy.hpp"
5 #include <list>

```

```

6 #include <memory>
16 class FreezingTower : public Tower{
17 public:
23     FreezingTower(sf::Vector2f);
30     void update(std::list<std::shared_ptr<Enemy>> &enemies, sf::Time time) override;
36     Projectile* shoot() override;
41     void upgradeTower() override;
42 private:
50     std::list<std::shared_ptr<Enemy>> lockedEnemies_;
58     float slowCoefficient_ = 0.2;
59
60 };
61 #endif //FREEZING_TOWER

```

## 7.9 game.hpp

```

1 #ifndef GAME_HPP
2 #define GAME_HPP
3
4 #include <SFML/Graphics.hpp>
5 #include <list>
6 #include "tower.hpp"
7 #include "path.hpp"
8 #include "enemy.hpp"
9 #include "projectile.hpp"
10 #include "resource_container.hpp"
11 #include "player.hpp"
12 #include <memory> //for shared_ptr
13 #include "bulletTower.hpp"
14 #include "button.hpp"
15 #include "map.hpp"
16 #include "missileProjectile.hpp"
17 #include "menu.hpp"
18 #include <vector>
19 #include "levelManager.hpp"
20 #include "explosion.hpp"
21
22 class Menu;
23 // Class for running the game logic
24
25 class Game {
26
27     friend class Tower;
28     friend class BulletTower;
29     friend class BombTower;
30     friend class MissileTower;
31     friend class FreezingTower;
32     friend class BombProjectile;
33     friend class BulletProjectile;
34     friend class MissileProjectile;
35     friend class PoisonTower;
36     friend class Menu;
37     friend class LevelManager;
38
39 public:
40     Map map;
41     Game();
42
43     void run();
44
45     ~Game() {
46         enemies_.clear();
47         for(auto i : projectiles_){
48             delete i;
49         }
50         projectiles_.clear();
51         towers_.clear();
52         // Menus deleted by unique_ptr
53     }
54
55     path& getPath();
56 private:
57     void processEvents();
58
59     void update();
60
61     void render();
62
63     void loadTextures();

```

```

112
116     void createPath(); //this will create the path that the enemies will traverse (this should also be
    rendered visually in the game)
117
124     void checkTowers();
125
126
127     void testEnemy();
128     void testEnemySplit(sf::Vector2f position, std::queue<sf::Vector2f> waypoints);
129
136     void updateMenus();
137
138     //adding a function to return the elapsed time
139     sf::Time getTime() const;
140     //I am adding a clock and time functionality that will need to be used for enemy movement and
    updating and other game logic
141     sf::Clock clock_;
142     sf::Time time_;
143     sf::RenderWindow window_;
144
145     std::list<std::shared_ptr<Tower>> towers_;
146     std::list<std::shared_ptr<Enemy>> enemies_;
147     std::list<Projectile*> projectiles_;
148     std::list<Explosion*> explosions_;
149     path path_;
150
151     bool dragged_;
152     bool paused_;
153     bool isGameOver_=false;
154     bool isGameFinished_ = false;
155     sf::Font font_;
156     sf::Text gameOverText;
157     sf::Text gameFinishedText;
158     sf::Sprite castle_sprite_;
159
160     std::unique_ptr<Menu> shop_;
161     std::unique_ptr<Menu> alternativeMenu_;
162     std::shared_ptr<Tower> activeTower_;
163     bool menuInactive = false;
164
165     ResourceContainer<Textures::TowerID, sf::Texture> tower_textures_;
166     ResourceContainer<Textures::EnemyID, sf::Texture> enemy_textures_;
167     ResourceContainer<Textures::ProjectileID, sf::Texture> projectile_textures_;
168     ResourceContainer<Textures::Various, sf::Texture> various_textures_;
169
170     Player player_;
171
172     LevelManager levelManager_;
173 };
174
175 #endif

```

## 7.10 levelManager.hpp

```

1 #ifndef LEVELMANAGER
2 #define LEVELMANAGER
3
4 #pragma once
5
6 #include <iostream>
7 #include <string>
8 #include <vector>
9 #include <map>
10 #include <variant>
11 #include <fstream>
12 #include <sstream>
13 #include <random>
14
15 #include "enemy.hpp"
16 #include "path.hpp"
17
18 class Game;
19
23 class LevelManager {
24
25     public:
29     using variantData = std::variant<int, float, std::vector<int>>;
30
41     LevelManager(const std::string& src, path& path, Game& game, Player& player) : src_(src),
    path_(path), game_(game), player_(player) {
42         readLevels();
43
44         currLevel_ = 0;

```

```

45         waitTime_ = 0;
46         levelTotal_ = levelSpecs_.size();
47     }
48     ~LevelManager() {}
49
53     int getCurrentLevel() const;
54
58     int getLevelTotal() const;
59
67     void update();
68
74     bool readingSuccessfull();
75
76     private:
77
86     void readLevels();
87
94     void initiateEnemies();
95
96
109     std::vector<std::map<std::string, variantData> levelSpecs_;
110
111     int currLevel_;
112     const std::string& src_;
113     bool readingSuccess_;
114     int levelTotal_;
115     float waitTime_;
116
117     path& path_;
118     Game& game_;
119     Player& player_;
120 };
121
122 #endif

```

## 7.11 map.hpp

```

1 #ifndef MAP_HPP
2 #define MAP_HPP
3
4 #include <SFML/Graphics.hpp>
5 #include <memory>
6 #include <string>
7 #include <vector>
8 #include "tower.hpp"
9
10 class Tower; // Forward declaration
11
12 class Map : public sf::Drawable, public sf::Transformable {
13 public:
14     sf::Texture texture;
15     sf::Sprite background;
16
17     void loadMap(const std::string& fileName);
18
19     std::vector<sf::FloatRect> unBuildable;
20
21 private:
22     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
23 };
24
25
26 #endif // MAP_HPP

```

## 7.12 menu.hpp

```

1 #ifndef MENU
2 #define MENU
3 #include <SFML/Graphics.hpp>
4 #include <list>
5 #include "button.hpp"
6 #include "game.hpp"
7 #include "tower.hpp"
8
9 // These are used in createMenu()
10 // the enum determines what type of menu is created:
11 // Which buttons are added etc.
12 enum class MenuType{
13     Shop,

```

```

14     Upgrade,
15     Begin,
16     Level
17 };
22 class Menu {
23 public:
29     void draw(sf::RenderWindow& window);
30
39     void checkButtons(Game* game);
40
47     void createMenu(MenuType menu, Game* game);
48
56     void update(Player& player);
57
69     void drag(Game* game);
70
76     void drawRange(Game* game);
77 private:
78
85     void newTower(std::shared_ptr<Tower> tower, Game* game);
86
93     bool canBePlaced(Game* game);
94
95     std::list<Button> buttons_;
96     std::vector<sf::Text> texts_;
97     sf::RectangleShape bg_;
98 };
99
100 #endif

```

## 7.13 missileProjectile.hpp

```

1 #ifndef MISSILE_PROJECTILE
2 #define MISSILE_PROJECTILE
3
4 #include "projectile.hpp"
5
9 class MissileProjectile : public Projectile
10 {
11 private:
12     std::shared_ptr<Enemy> targetEnemy_;
13
14 public:
21     MissileProjectile(sf::Vector2f position, int damage, std::shared_ptr<Enemy> targetEnemy)
22         : Projectile(sf::Vector2f(0,0), position, damage, 280.f, "missile", 400), targetEnemy_(targetEnemy)
23     {}
31     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) override;
32
41     void update(Game& game) override;
42
46     Textures::ProjectileID textureType() override { return Textures::Missile; }
47 };
48
49
50 #endif

```

## 7.14 missileTower.hpp

```

1 #ifndef MISSILE_TOWER
2 #define MISSILE_TOWER
3 #include "tower.hpp"
4 #include "missileProjectile.hpp"
12 class MissileTower : public Tower {
13 public:
18     MissileTower(sf::Vector2f);
24     MissileProjectile* shoot() override;
25 };
26 #endif

```

## 7.15 path.hpp

```

1 #ifndef PATH_HPP
2 #define PATH_HPP
3 #include <queue>

```



```

4 #include <SFML/System/Vector2.hpp>
5 #include <SFML/Graphics.hpp>
6 #include <vector>
7 #include <random>
8
9 class path {
10     friend class enemy;
11 public:
12     path(const std::string& src) : src_(src) {
13         readPath();
14
15         std::random_device rd;
16         std::uniform_int_distribution<int> range(0, paths_.size()-1);
17
18         auto gamePath = paths_[range(rd)];
19
20         for(const auto& point: gamePath){
21             addWaypoint(point);
22         }
23     }
24
25     ~path() {}
26
27     void readPath();
28
29     bool readingSuccessfull();
30
31     void addWaypoint(const sf::Vector2f& point);
32
33     std::queue<sf::Vector2f> getWaypoints() const;
34
35     void makeUnBuildablePath();
36     static const float width;
37     std::queue<sf::Vector2f> waypoints_;
38     std::vector <sf::Vector2f> wayPoints;
39     std::vector <sf::FloatRect> unBuildable;
40
41     std::vector<std::vector<sf::Vector2f>> paths_;
42
43 private:
44     const std::string& src_;
45     bool readingSuccess_;
46 };
47
48 #endif

```

## 7.16 player.hpp

```

1 #ifndef PLAYER
2 #define PLAYER
3
4 #include <string>
5 #include <list>
6 #include "enemy.hpp"
7 #include "tower.hpp"
8 #include <SFML/System/Vector2.hpp>
9 #include <SFML/Graphics/Transformable.hpp>
10 #include <memory>
11 #include "resource_container.hpp"
12
13 class Tower;
14 class Enemy;
15
16 class Player : public sf::Sprite
17 {
18     private:
19         int hp_;
20         int wallet_;
21         int level_;
22
23     public:
24         Player() : hp_(500), wallet_(1000), level_(0){}
25
26         ~Player() {}
27
28         int getWallet() const;
29
30         int getHP() const;
31
32         int getLevel() const;
33 };
34
35 #endif

```

```

58     void levelUp();
59
60     void addMoney(int amount);
61
62     void removeMoney(int cost);
63
64     void removeHP(int amount);
65 };
66
67 #endif

```

## 7.17 poisonTower.hpp

```

1  #ifndef POISON_TOWER
2  #define POISON_TOWER
3  #include "tower.hpp"
4  #include "enemy.hpp"
5  #include <list>
6  #include <memory>
7
8  class PoisonTower : public Tower{
9  public:
10     PoisonTower(sf::Vector2f);
11     void update(std::list<std::shared_ptr<Enemy>> &enemies, sf::Time time) override;
12     Projectile* shoot() override;
13 private:
14     std::list<std::shared_ptr<Enemy>> lockedEnemies_;
15 };
16
17 #endif

```

## 7.18 projectile.hpp

```

1  #ifndef PROJECTILE
2  #define PROJECTILE
3
4  #include "tower.hpp"
5  #include "player.hpp"
6  #include "enemy.hpp"
7  #include "resource_container.hpp"
8  #include <SFML/System/Vector2.hpp>
9  #include <SFML/Graphics/Transformable.hpp>
10 #include <SFML/Graphics.hpp>
11 #include <memory>
12 #include <iostream>
13
14 class Game;
15 class Enemy;
16
17 class Projectile : public sf::Sprite
18 {
19 private:
20     float speed_;
21     std::string type_;
22     int damage_;
23     sf::Vector2f position_; // of tower that created
24     float maxDistance_;
25     sf::Vector2f shootDirection_;
26     bool isDestroyed_;
27
28 public:
29     Projectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed,
30               std::string type, float maxDistance)
31         : shootDirection_(shootDirection), position_(position), damage_(damage), speed_(speed),
32           type_(type), maxDistance_(maxDistance),
33           isDestroyed_(false){
34         this->setPosition(position_);
35     }
36
37     virtual ~Projectile() {}
38
39     float getSpeed() const;
40
41     const std::string& getType() const;
42
43     int getDamage() const;
44
45     sf::Vector2f getShootDir() const;
46     //sf::Vector2f getVelocity() const;
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

```

```

78     void destroy();
79
83     bool isDestroyed();
84
89     bool distToTower();
90
94     virtual bool hasHitEnemy(std::shared_ptr<Enemy>&) = 0;
95
99     virtual void update(Game&) = 0;
100
105     virtual Textures::ProjectileID textureType() = 0;
106 };
107 #endif
108

```

## 7.19 resource\_container.hpp

```

1 #ifndef RESOURCE_CONTAINER
2 #define RESOURCE_CONTAINER
3 #include <SFML/Graphics.hpp>
4 #include <string>
5 #include <memory>
6
7 // Enums for different textures
8 namespace Textures{
9
10     // NOTE: these could also be stored in one big enum...
11     enum TowerID {BulletTower, BombTower, MissileTower, FreezingTower, PoisonTower};
12     enum EnemyID {Enemy1, Enemy2, Enemy3, Enemy4, Enemy5};
13     enum ProjectileID{Bullet, Bomb, Missile};
14     enum Various {Pause, Castle, Dirt, Upgrade, Sell, Continue};
15 }
16
23 template <typename T_enum, typename T_resource>
24 class ResourceContainer {
25 public:
26
33     void load(T_enum type, std::string filename){
34         std::unique_ptr<T_resource> resource(new T_resource());
35
36         if (!resource->loadFromFile(filename)){
37             //TODO: Handle texture loading error
38         }
39         // The function move should avoid creating a copy of the object resource, when inserting it into
40         the map
41         resources_.insert(std::make_pair(type, std::move(resource)));
42     }
43
50     T_resource& get(T_enum type) const {
51         auto wanted = resources_.find(type);
52         return *wanted->second;
53     }
54
55 private:
56     std::map<T_enum, std::unique_ptr<T_resource>> resources_;
57
58 };
59
60 };
61
62 #endif

```

## 7.20 tower.hpp

```

1 #ifndef TOWER_H
2 #define TOWER_H
3 #include <string>
4 #include <array>
5 #include <SFML/System/Vector2.hpp>
6 #include <SFML/System/Clock.hpp>
7 #include <SFML/Graphics.hpp>
8 #include "projectile.hpp"
9 #include "enemy.hpp"
10 #include <memory>
11
12
13 class Projectile;
14 class Tower : public sf::Sprite {
15 public:

```

```

26     Tower(sf::Vector2f position, const std::string& type, int baseCost, float range, sf::Time fireRate,
27           int damage, int upgradeCost);
28     const std::string& getType() const {return type_;}
29     const int getBaseCost() const {return baseCost_;}
30     sf::Time getFireRate() const {return fireRate_;}
31     const float getRange() const {return range_;}
32     int getDamage() const {return damage_;}
33     std::shared_ptr<Enemy> getLockedEnemy() const {return lockedEnemy_;}
34     bool isMaxLevelReached() const {return maxLevelReached_};
35     int getCurrentLvl() const {return currentLvl_;}
36     const int getUpgradeCost() const {return upgradeCost_};
37     sf::Time getFireTimer() {return fireTimer_;}
38     bool enemyWithinRange(std::shared_ptr<Enemy> enemy);
44     virtual Projectile* shoot() = 0;
52     virtual void upgradeTower();
57     virtual void update(std::list<std::shared_ptr<Enemy> &enemies, sf::Time time);
58     void updateFireTimer(sf::Time &dt);
59     void setLevel(int level) {currentLvl_ = level;}
60     void setMaxLevelFlag() {maxLevelReached_ = true;}
61     void setLockedEnemy(std::shared_ptr<Enemy> enemy) {lockedEnemy_ = enemy;}
62     void resetFireTimer() {fireTimer_ = sf::Time::Zero;}
75 private:
76     const std::string type_;
77     const int baseCost_;
78     const float range_;
79     int damage_;
80     int currentLvl_;
81     const int upgradeCost_;
82     std::shared_ptr<Enemy> lockedEnemy_;
83     sf::Time fireTimer_;
84     sf::Time fireRate_;
85     bool maxLevelReached_;
86 };
87 #endif //TOWER_H

```

# Index

- addMoney
  - Player, [50](#)
- addWaypoint
  - path, [48](#)
- BombProjectile, [17](#)
- BombTower, [17](#)
  - BombTower, [18](#)
  - shoot, [18](#)
  - update, [18](#)
- BulletProjectile, [19](#)
  - hasHitEnemy, [19](#)
  - textureType, [20](#)
  - update, [20](#)
- BulletTower, [20](#)
  - BulletTower, [21](#)
  - shoot, [21](#)
- Button, [22](#)
  - Button, [22](#)
  - isClicked, [23](#)
- canBePlaced
  - Menu, [41](#)
- checkButtons
  - Menu, [41](#)
- checkTowers
  - Game, [34](#)
- createMenu
  - Menu, [41](#)
- dead
  - Enemy, [25](#)
- destroy
  - Projectile, [55](#)
- distToTower
  - Projectile, [55](#)
- drag
  - Menu, [42](#)
- draw
  - Menu, [42](#)
- drawRange
  - Menu, [42](#)
- Enemy, [23](#)
  - dead, [25](#)
  - Enemy, [24](#)
  - getCenter, [25](#)
  - getHealthText, [25](#)
  - getLocation, [26](#)
  - getMoney, [26](#)
  - getWaypoints, [26](#)
  - hp, [26](#)
  - initialHp, [26](#)
  - isWaypointPassed, [27](#)
  - poisonStatus, [27](#)
  - setVelocity, [27](#)
  - slowedStatus, [27](#)
  - speed, [27](#)
  - type, [28](#)
  - update, [28](#)
- enemyWithinRange
  - Tower, [61](#)
- Explosion, [28](#)
  - Explosion, [29](#)
  - update, [29](#)
- FreezingTower, [30](#)
  - FreezingTower, [30](#)
  - lockedEnemies\_, [32](#)
  - shoot, [31](#)
  - slowCoefficient\_, [32](#)
  - update, [31](#)
  - upgradeTower, [31](#)
- Game, [32](#)
  - checkTowers, [34](#)
  - getPath, [34](#)
  - processEvents, [35](#)
  - render, [35](#)
  - run, [35](#)
  - update, [35](#)
  - updateMenus, [36](#)
- get
  - ResourceContainer< T\_enum, T\_resource >, [58](#)
- getCenter
  - Enemy, [25](#)
- getCurrentLevel
  - LevelManager, [37](#)
- getDamage
  - Projectile, [56](#)
- getHealthText
  - Enemy, [25](#)
- getHP
  - Player, [50](#)
- getLevel
  - Player, [50](#)
- getLevelTotal
  - LevelManager, [38](#)
- getLocation
  - Enemy, [26](#)

- getMoney
  - Enemy, 26
- getPath
  - Game, 34
- getShootDir
  - Projectile, 56
- getSpeed
  - Projectile, 56
- getType
  - Projectile, 56
- getWallet
  - Player, 51
- getWaypoints
  - Enemy, 26
- hasHitEnemy
  - BulletProjectile, 19
  - MissileProjectile, 44
  - Projectile, 56
- hp
  - Enemy, 26
- initialHp
  - Enemy, 26
- isClicked
  - Button, 23
- isWaypointPassed
  - Enemy, 27
- LevelManager, 36
  - getCurrentLevel, 37
  - getLevelTotal, 38
  - LevelManager, 37
  - levelSpecs\_, 39
  - readingSuccessfull, 38
  - readLevels, 38
  - update, 38
- levelSpecs\_
  - LevelManager, 39
- load
  - ResourceContainer< T\_enum, T\_resource >, 58
- lockedEnemies\_
  - FreezingTower, 32
  - PoisonTower, 53
- Map, 39
- Menu, 40
  - canBePlaced, 41
  - checkButtons, 41
  - createMenu, 41
  - drag, 42
  - draw, 42
  - drawRange, 42
  - newTower, 43
  - update, 43
- MissileProjectile, 43
  - hasHitEnemy, 44
  - MissileProjectile, 44
  - textureType, 45
  - update, 45
- MissileTower, 46
  - MissileTower, 46
  - shoot, 47
- newTower
  - Menu, 43
- path, 47
  - addWaypoint, 48
  - path, 48
  - readingSuccessfull, 48
  - readPath, 48
- Player, 49
  - addMoney, 50
  - getHP, 50
  - getLevel, 50
  - getWallet, 51
  - Player, 50
  - removeHP, 51
  - removeMoney, 51
- poisonStatus
  - Enemy, 27
- PoisonTower, 52
  - lockedEnemies\_, 53
  - PoisonTower, 52
  - shoot, 53
  - update, 53
- processEvents
  - Game, 35
- Projectile, 54
  - destroy, 55
  - distToTower, 55
  - getDamage, 56
  - getShootDir, 56
  - getSpeed, 56
  - getType, 56
  - hasHitEnemy, 56
  - Projectile, 55
  - textureType, 57
  - update, 57
- readingSuccessfull
  - LevelManager, 38
  - path, 48
- readLevels
  - LevelManager, 38
- readPath
  - path, 48
- removeHP
  - Player, 51
- removeMoney
  - Player, 51
- render
  - Game, 35
- ResourceContainer< T\_enum, T\_resource >, 57
  - get, 58
  - load, 58
- run

- Game, [35](#)
- setVelocity
  - Enemy, [27](#)
- shoot
  - BombTower, [18](#)
  - BulletTower, [21](#)
  - FreezingTower, [31](#)
  - MissileTower, [47](#)
  - PoisonTower, [53](#)
  - Tower, [61](#)
- slowCoefficient\_
  - FreezingTower, [32](#)
- slowedStatus
  - Enemy, [27](#)
- speed
  - Enemy, [27](#)
- src/bombProjectile.hpp, [65](#)
- src/bombTower.hpp, [65](#)
- src/bulletProjectile.hpp, [65](#)
- src/bulletTower.hpp, [66](#)
- src/button.hpp, [66](#)
- src/enemy.hpp, [66](#)
- src/explosion.hpp, [68](#)
- src/freezingTower.hpp, [68](#)
- src/game.hpp, [69](#)
- src/levelManager.hpp, [70](#)
- src/map.hpp, [71](#)
- src/menu.hpp, [71](#)
- src/missileProjectile.hpp, [72](#)
- src/missileTower.hpp, [72](#)
- src/path.hpp, [72](#)
- src/player.hpp, [73](#)
- src/poisonTower.hpp, [74](#)
- src/projectile.hpp, [74](#)
- src/resource\_container.hpp, [75](#)
- src/tower.hpp, [75](#)
- textureType
  - BulletProjectile, [20](#)
  - MissileProjectile, [45](#)
  - Projectile, [57](#)
- Tower, [59](#)
  - enemyWithinRange, [61](#)
  - shoot, [61](#)
  - Tower, [60](#)
  - type\_, [62](#)
  - update, [61](#)
  - updateFireTimer, [62](#)
  - upgradeTower, [62](#)
- type
  - Enemy, [28](#)
- type\_
  - Tower, [62](#)
- update
  - BombTower, [18](#)
  - BulletProjectile, [20](#)
  - Enemy, [28](#)
  - Explosion, [29](#)
  - FreezingTower, [31](#)
  - Game, [35](#)
  - LevelManager, [38](#)
  - Menu, [43](#)
  - MissileProjectile, [45](#)
  - PoisonTower, [53](#)
  - Projectile, [57](#)
  - Tower, [61](#)
  - updateFireTimer
    - Tower, [62](#)
  - updateMenus
    - Game, [36](#)
  - upgradeTower
    - FreezingTower, [31](#)
    - Tower, [62](#)