Druidic Defence

Generated by Doxygen 1.9.3

Chapter 1

Project Documentation

1.1 Group Members

- Julius Järvilinna 596174
- Tousif Zaman 980285
- Tomi Mikkola 793155
- Konsta Kemppainen 1008250

1.2 Overview

We made a pretty conventional Tower Defense (TD) game with 6 unique tower types and 4 different enemies, 2 maps, and 10 enemy rounds. In our TD game, towers can be bought, sold and upgraded during or in-between rounds. Each tower has 3 upgrades with the third upgrade giving the tower some special ability. Different enemies have different health and movement speed values. We have two different damage types, some towers being more effective against certain enemy types. We also have a boss enemy that is immune to slowing towers. We have nice background music which sound level can be changed in the settings. Our level map consists of a background image, where the enemies path consists of predetermined points on the map. Towers can't be placed on the path of the enemies.

Maps also have obstacles such as trees and rocks that towers cannot be placed on. We thought also of making the map consist of tiles, but decided we want rather to make it a background image with an array of 2D points to describe the enemy path, because we thought it would look cooler.

The in-game user interface allows players to buy and place towers, hover over towers in the shop to gain more information on them and start rounds. The UI shows how much money and lives the player has and how much towers cost. Towers placed on the map can be clicked on to show their range and open up a menu that lets the player upgrade and sell the tower. We also added targeting options such as those in Bloons TD games. Killed enemies drop nature's essence, which is used as currency for buying and upgrading towers. Enemies with more HP drop more money. If an enemy gets past the player's defenses, the player will lose lives based on the amount of money the enemy would have dropped. Maps and enemy rounds are read from files.

Towers can be clicked after placement to open a menu in which you can upgrade, sell and change targeting options of the tower. Towers start at tier 1 and can be upgraded thrice. The first two upgrades increase some of the basic stats: damage, rate of fire and range. Most tier 4 upgrades are special and change the tower in some way. In the same menu a tower can be set to target the first, last, closest or strongest enemy within its range.

There are four enemy types. The cockroach is a basic enemy. Flies are faster than cockroaches. Beetles are armored and have a lot of health. Dragonflies are armored bosses that have a lot of health and are immune to slowing effects. Armored enemies have a yellow health bar. They have a lot of health, but take extra damage from certain sources such as the coconut cannon and sniper towers.

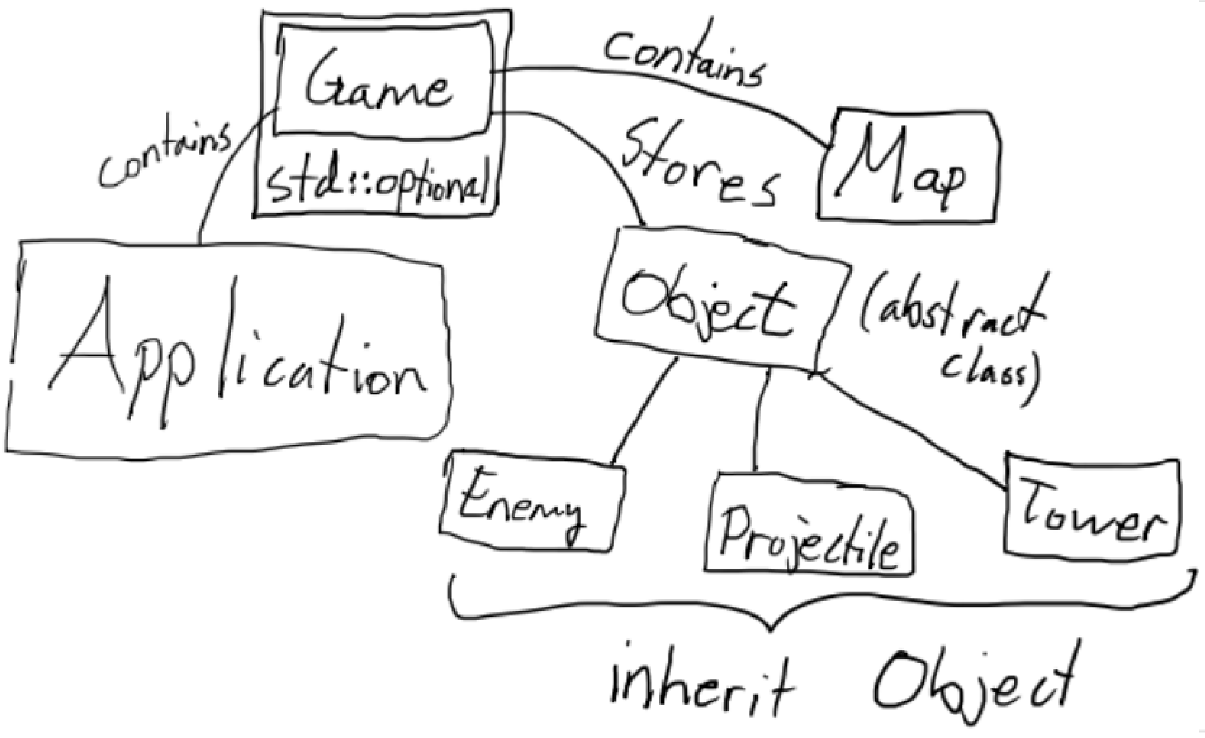
1.3 Game Play

The game opens in the main menu, from which the player can start the game, open the options menu or quit the game. In the options menu sound volume and whether rounds start automatically can be changed. Starting the game puts the player in a map select screen where they can choose between two different maps.

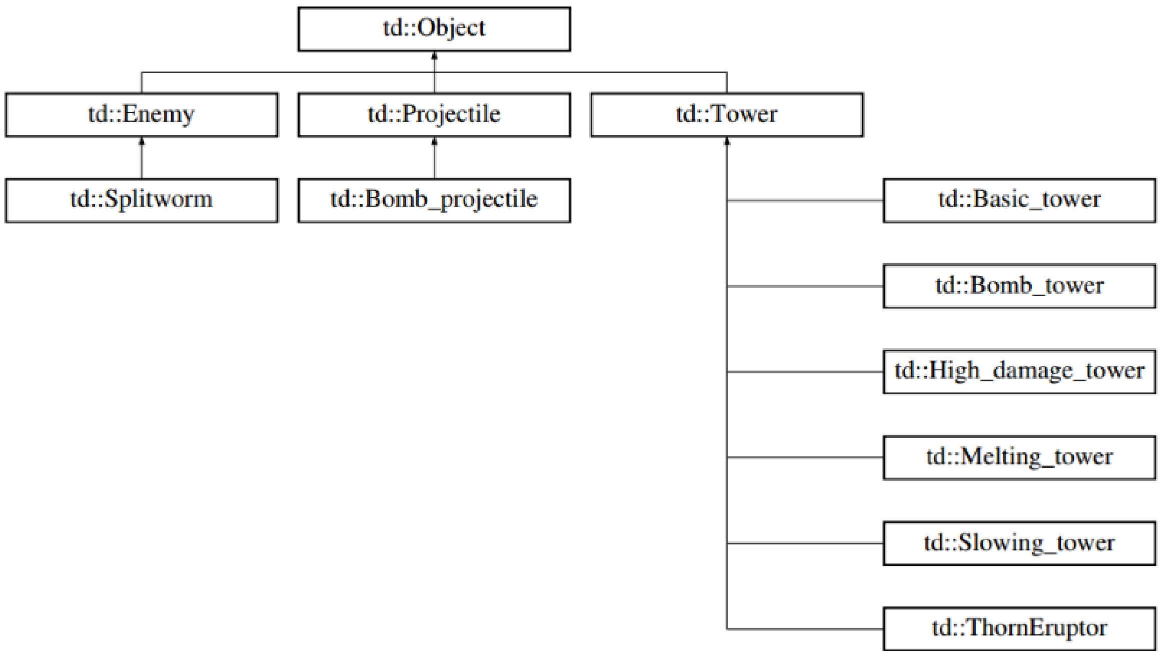
Once a map is selected the game starts. Rounds of enemies rush through the path on the map and the player must stop them from getting to the end of the path by placing towers on the map. The game ends once all rounds are beaten or the player runs out of health. There are 6 different towers in the shop on the right, more info on the towers can be gained by hovering over them in the GUI. The game can be paused from the top right. Towers are bought and placed by clicking on one of the shop icons and then clicking on the game map in the desired position. Towers can be clicked on after placement to open the upgrading menu.

1.4 Software Structure

The below diagram representes the high level design philosophy of the game.



The below diagram shows the inheritance relationship between the game objects.



1.5 Data Structure Decisions

Some of the data structure decisions we took in the implementation are highlighted below:

- Game objects are stored in lists because removal is `O(1)`
- Collisions are handled using maps because accessing a map is `O(log n)`
- We preferred using `std::optional` over `nullptr` to enforce error handling and to avoid null chaos
- We used `unsigned int` wherever applicable (as opposed to bare `int`)
- We favor `const` references instead of copying wherever applicable (for example `const std::string&`)
- `sf::vector2f` are always passed in by copy because they are just 8 bytes and copying them is faster

1.6 Instructions

First you need to clone the repository and download all the required dependencies (git submodules). This can be achieved with Git by doing the following:

1. Open your terminal in the directory where you want to download the project.
2. Run `git clone --recurse-submodules git@version.aalto.fi:cpp-autumun-2021/tower-defens` `git` in the terminal.

The rest is platform-specific. Please read below to see instructions for your platform.

1.7 How to compile the program

1.7.1 Windows

Tested on Windows 10.

1. Install CMake from the `CMake website`.
2. Install `Visual Studio 2022`.
3. Install the `Desktop development with C++` workload.
4. Make a build folder in the project directory.
5. Once you have a terminal open in the build folder, run `cmake ..`
6. If the files are created successfully, open `TowerDefense5.sln` in Visual Studio 2022.
7. Build and run the project in Visual Studio 2022.

1.7.2 Linux

Tested on Ubuntu 18.04.

1. Run `sudo apt update` in the terminal.
2. Install toolchain (GCC, make, etc.) by running `sudo apt install build-essential` in the terminal.
3. Install CMake by running `sudo apt install cmake` in the terminal.
4. Open a terminal in the project directory.
5. `mkdir build` - make a `build` directory inside the project.
6. `cd build` - enter the build directory.
7. `cmake ..` - run CMake and generate a makefile inside `build`.
8. `make` - use the makefile to build the project.
9. `./TowerDefense5` - finally, run the executable generated inside `bin`.

1.8 Testing

We wrote our own tests for collision detections; otherwise, we tested most aspects of our game visually by running TowerDefense5.exe and checking for problems. We use `GoogleTest` to write unit tests. The `CMakeLists.txt` takes care of installing the necessary dependencies and generates an executable for all unit test sources present inside `tests` directory. Run this executable to see the status of unit tests (passed/failed).

1.8.1 Steps to generate text executable and run it

```
cd tower-defense-5          # enter project root directory
mkdir build && cd build     # make a build directory
cmake ..                    # run CMake
make                        # make tests along with sources
../bin/TowerDefense5_tests  # run tests executable from executables directory
```

1.9 Work log

Tomi took care of the artistic side of our project, like designing how the enemies and towers look like and did the GUI of our game. Tomi also worked on the application class of our game. Konsta worked with CMake at the start of our project. Later Konsta worked on Map.cpp and Game.cpp. Tousif also worked with CMake, designed abstract classes of our game, implemented the collision detection algorithms with unit tests. Julius was responsible for towers and enemies planning and he implemented tower and projectile subclasses. Julius also made some functions into tower, projectile and enemy abstract classes. Everyone was involved in our weekly meetings for most weeks and took part in planning and discussing the game features. Overall team workload was weighted more at the end of the project, when the deadline was nearing. It took a while until the project picked up steam because CMake was a little problematic (on Windows especially).

Detailed weekly work done was as follows:

1.9.1 Week 1 (1.11-8.11)

- Tomi did Trello board for our team
- Konsta researched CMake and tried to get it to work on Windows using the Visual Studio compiler. He also imported the external libraries as git submodules. Getting them to link properly was surprisingly difficult.
- Julius tried to get the project working on school ssh connection
- Tousif created the basic structure of the project, added CMake support

Time used for project on week 1 per team member:

- Julius 6h
- Tousif 8h
- Konsta 12h
- Tomi 4h

1.9.2 Week 2 (9.11-15.11)

- Julius planned 5 basic towers and tried to get the code to compile with VScode
- Tousif and Konsta worked on the CMakeLists and finally got it working.
- Tousif and Konsta helped Julius and Tomi to get the code to compile
- Use of Git was discussed and teached to Julius and Tomi, whom had only little previous experience with it

Time used for project on week 2 per team member:

- Julius 6h
- Tousif 8h
- Konsta 8h
- Tomi 5h

1.9.3 Week 3 (16.11-22.11)

- Tomi did the artwork for many towers, enemies and a map
- Julius designed several enemies
- Tousif worked on abstract classes and added googletest support for the project
- Konsta improved the README (instructions for how to compile the code on Windows) and figured out how to make it automatically generate solution filters for Visual Studio.

Time used for project on week 3 per team member:

- Julius 4h
- Tousif 8h
- Konsta 2h
- Tomi 10h

1.9.4 Week 4 (23.11-29.11)

- Tomi worked on more artwork
- Julius and Tousif solved problems with getting the tests to work on Windows in VScode and with Mingw
- Julius worked on doing tower implementations
- Konsta cleaned up some code and added a missing getter.
- Tousif finished up abstract classes, added basic unit tests for it

Time used for project on week 4 per team member:

- Julius 15h
- Tousif 15h
- Konsta 2h
- Tomi 4h

1.9.5 Week 5 (30.11-6.12)

- Tomi worked on the GUI and application class
- Julius worked on towers and enemies
- Tousif added clang-format support to format the code, worked on collision detection math involving circles and polygons on paper and started the implementation
- Konsta cleaned up a lot of code and worked on Game.cpp.

Time used for project on week 5 per team member:

- Julius 20h
- Tousif 25h
- Konsta 8h
- Tomi 40h

1.9.6 Week 6 (7.12-12.12)

- Tomi finished off GUI, made all missing artwork and helped with the game class
- Julius finished his implementation of towers and enemies and worked on project documentation
- Tousif implemented and fixed bugs in collision detection functions and added them to Game.
- Konsta worked on Game.cpp (added a function for loading in enemies and a function for loading in rounds) and implemented the Update method for Game. He also cleaned up a lot of code and added.

Time used for project on week 6 per team member:

- Julius 8h
- Tousif 25h
- Konsta 25h
- Tomi 35h

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

Chapter 6

Namespace Documentation

6.1 td::types Namespace Reference

Define types useful to our game.

Typedefs

- using **Position** = sf::Vector2f

  The type that stores the position of any game object.
- using **Time** = sf::Time

  The type that stores the time.
- using **BlockedRegion** = sf::ConvexShape

  Represents a region where towers cannot be placed.

Enumerations

- enum **Targeting** {
  kFirst , **kLast** , **kClose** , **kStrong** ,
  kArea }
- enum **AppState** {
  kMainMenu , **kOptions** , **kMapSelect** , **kGame** ,
  kPause , **kUpgrade** }

6.1.1 Detailed Description

Define types useful to our game.

Chapter 7

Class Documentation

7.1 td::Application Class Reference

Application class represents the app that runs the game. It takes care of the game's gui and rendering.

```
#include <application.hpp>
```

Public Member Functions

- **Application** ()

  Application constructor.
- int **run** ()

  Runs the application.
- void **LoadTextures** ()

  Loads all of the game's textures into textures_.
- void **LaunchMainMenuGui** ()

  Creates TGUI widgets for the main menu.
- void **LaunchMapSelectGui** ()

  Creates TGUI widgets for the map selection screen.
- void **LaunchGameGui** ()

  Creates TGUI widgets for the game gui.
- void **LaunchOptionsGui** ()

  Creates TGUI widgets for the options menu.
- void **LaunchPauseGui** ()

  Creates TGUI widgets for the pause menu.
- void **LaunchUpgradeGui** ()

  Creates TGUI widgets for tower upgrading.
- void **LaunchGame** (const std::string &map_name)

  Creates TGUI widgets for the main menu.
- void **CloseGame** ()

  Closes the game and returns to main menu. Takes care of deletion of objects.
- void **HandleMainMenu** ()

  Handles events and draws sprites when in the main menu.
- void **HandleMapSelect** ()

  Handles events and draws sprites when in map select.

- void **HandleGame** ()

  Handles events and draws sprites when in-game.
- void **HandleOptions** ()

  Handles events and draws sprites when in the options menu.
- void **HandlePause** ()

  Handles events and draws sprites when in the pause menu.
- void **HandleUpgrade** ()

  Handles events and draws sprites when upgrading a tower.
- void **HandleMainMenuGui** ()

  Handles TGUI signals of the main menu gui.
- void **HandleMapSelectGui** ()

  Handles TGUI signals of the map select gui.
- void **HandleGameGui** ()

  Handles TGUI signals of the game gui.
- void **HandleOptionsGui** ()

  Handles TGUI signals of the options menu gui.
- void **HandlePauseGui** ()

  Handles TGUI signals of the pause menu gui.
- void **HandleUpgradeGui** ()

  Handles TGUI signals of the upgrade menu gui.
- void **TargetingSwitchRight** ()

  Gui helper function that changes the targeting option of the currently selected tower "rightwards".
- void **TargetingSwitchLeft** ()

  Gui helper function that changes the targeting option of the currently selected tower "leftwards".
- void **DrawGameElements** ()

  Draws towers, projectiles, enemies and their health bars.
- void **DrawShopElements** ()

  Draws the elements of game gui that go over TGUI elements.
- void ScaleSprite (sf::Transformable &sprite)

  scales an sf::Sprite to have correct proportions relative to the window called once for every sprite
- void StyleButtonBrown (tgui::Button::Ptr button)

  Changes a tgui::Button to match the game's artstyle.

7.1.1 Detailed Description

Application class represents the app that runs the game. It takes care of the game's gui and rendering.

7.1.2 Member Function Documentation

7.1.2.1 ScaleSprite()

```
void td::Application::ScaleSprite (
             sf::Transformable & sprite )
```

scales an sf::Sprite to have correct proportions relative to the window called once for every sprite

Parameters

sprite	Sprite that is scaled

7.1.2.2 StyleButtonBrown()

```
void td::Application::StyleButtonBrown (
            tgui::Button::Ptr button )
```

Changes a tgui::Button to match the game's artstyle.

Parameters

button	Button that gets changed

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

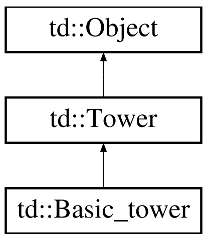
- tower-defense-5/include/application.hpp
- tower-defense-5/src/logic/application.cpp

7.2 td::Basic_tower Class Reference

//Implementation of our most basic tower type

```
#include <basic_tower.hpp>
```

Inheritance diagram for td::Basic_tower:



Public Member Functions

- Basic_tower (types::Position position, float rotation_angle=0.0f, sf::Texture ∗texture=nullptr, sf::Texture ∗texture_projectile=nullptr)

  Basic_tower constructor.
- void Upgrade ()

  Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

  Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &, std::list< Enemy > &enemies)

  Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.2.1 Detailed Description

//Implementation of our most basic tower type

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Basic_tower()

```
td::Basic_tower::Basic_tower (
            types::Position position,
            float rotation_angle = 0.0f,
            sf::Texture * texture = nullptr,
            sf::Texture * texture_projectile = nullptr )
```

Basic_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.2.3 Member Function Documentation

7.2.3.1 Shoot()

```
bool td::Basic_tower::Shoot (
            std::list< Projectile > & projectiles,
            std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

Reimplemented from td::Tower.

7.2.3.2 Update()

```
void td::Basic_tower::Update (
            types::Time dt,
            std::list< Enemy > & enemies,
            std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.2.3.3 Upgrade()

```
void td::Basic_tower::Upgrade ( )  [virtual]
```

Upgrades the tower once.

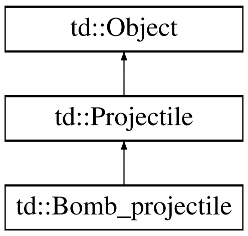
Reimplemented from td::Tower.

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

- tower-defense-5/include/basic_tower.hpp
- tower-defense-5/src/logic/towers/basic_tower.cpp

7.3 td::Bomb_projectile Class Reference

Inheritance diagram for td::Bomb_projectile:



Public Member Functions

- Bomb_projectile (types::Position position, float rotation_angle, int damage, sf::Texture ∗texture_projectile, sf::Texture ∗texture_explosion, float speed, float lifetime)

  Bomb_projectile constructor.

Additional Inherited Members

7.3.1 Constructor & Destructor Documentation

7.3.1.1 Bomb_projectile()

```
td::Bomb_projectile::Bomb_projectile (
            types::Position position,
            float rotation_angle,
            int damage,
            sf::Texture * texture_projectile,
            sf::Texture * texture_explosion,
            float speed,
            float lifetime )
```

Bomb_projectile constructor.

Parameters

position	Position of the projectile
rotation_angle	Orientation, in radians of the projectile
damage	Damage % of the projectile
texture_projectile	Pointer to the texture of the projectile
texture_explosion	Pointer to the texture of the explosion of the bomb

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

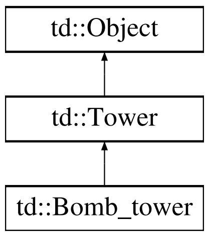
- tower-defense-5/include/bomb_projectile.hpp
- tower-defense-5/src/logic/projectiles/bomb_projectile.cpp

7.4 td::Bomb_tower Class Reference

//Implementation of bomb tower that shoots explosive bombs to enemies

`#include <bomb_tower.hpp>`

Inheritance diagram for td::Bomb_tower:



Public Member Functions

- Bomb_tower (sf::Vector2< float > position, float rotation_angle=0.0f, sf::Texture *texture=nullptr, sf::Texture *texture_projectile=nullptr, sf::Texture *texture_explosion=nullptr)

  Bomb_tower constructor.
- void Upgrade ()

  Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

  Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &, std::list< Enemy > &enemies)

  Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.4.1 Detailed Description

//Implementation of bomb tower that shoots explosive bombs to enemies

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Bomb_tower()

```
td::Bomb_tower::Bomb_tower (
             sf::Vector2< float > position,
             float rotation_angle = 0.0f,
             sf::Texture * texture = nullptr,
             sf::Texture * texture_projectile = nullptr,
             sf::Texture * texture_explosion = nullptr )
```

Bomb_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.4.3 Member Function Documentation

7.4.3.1 Shoot()

```
bool td::Bomb_tower::Shoot (
             std::list< Projectile > & projectiles,
             std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

Reimplemented from td::Tower.

7.4.3.2 Update()

```
void td::Bomb_tower::Update (
            types::Time dt,
            std::list< Enemy > & enemies,
            std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.4.3.3 Upgrade()

```
void td::Bomb_tower::Upgrade ( )  [virtual]
```

Upgrades the tower once.

Reimplemented from td::Tower.

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

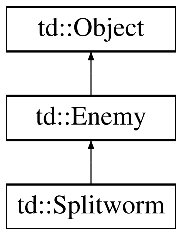
- tower-defense-5/include/bomb_tower.hpp
- tower-defense-5/src/logic/towers/bomb_tower.cpp

7.5 td::Enemy Class Reference

Enemy class represents the blueprint of a basic enemy in the game. The base enemy class can be derived further to create towers with special powers.

```
#include <enemy.hpp>
```

Inheritance diagram for td::Enemy:



Public Member Functions

- Enemy (types::Position position, float hitbox, sf::Texture ∗texture, float health=100.0f, int move_speed=1, int bounty=0, bool armored=false, float distance_moved=0.0f, unsigned int slowed_level=0)

  Enemy constructor.
- **Enemy** (const Enemy &enemy)
- virtual void **Update** (types::Time dt, const Game &game)
- void **Update** (types::Time dt, const std::vector< types::Position > &path)
- Enemy createBasicCockroach (types::Position startOfTheMap, sf::Texture ∗texture)

  Create basic cockroach at the start of the map.
- Enemy createFly (types::Position startOfTheMap, sf::Texture ∗texture)

  Create fly at the start of the map.
- Enemy createBeetle (types::Position startOfTheMap, sf::Texture ∗texture)

  Create beetle at the start of the map.
- Enemy createDragonfly (types::Position startOfTheMap, sf::Texture ∗texture)

  Create dragonfly (boss enemy) at the start of the map.
- float getHealth () const

  Get the remaining health of the enemy.
- void setHealth (float health_decrease)

  Set the health of the enemy.
- float getMaxHealth () const

  Get the maximum health of the enemy.
- int getMoveSpeed () const

  Get the movement speed of the enemy.
- int getBounty () const

  Get the bounty of the enemy.
- bool isArmored () const

  Status of enemy armor.
- bool isAtEndOfPath () const
- void **setDistanceMoved** (float distance)

  Set the total distance moved on the path by the enemy.
- float getDistanceMoved () const

  Get the distance enemy has travelled.
- void **setSlowedLevel** (unsigned int level)

  Set the slowed_level of the enemy.
- unsigned int getSlowedLevel () const

  Get the slowed level of the enemy.
- bool TakeDamage (float damage, bool is_armor_piercing)

  called when a tower or projectile wants to do damage to the enemy

Protected Attributes

- float **health_**

  Remaining health of the enemy.
- float **max_health_**

  Maximum health of the enemy.
- int **move_speed_**

  Movement speed of the enemy.
- int **bounty_**

  Bounty of the enemy.
- bool **armored_**

Status of enemy armor.

- float **distance_moved_**

  Total distance moved on the path by the enemy.

- unsigned int **slowed_level_**

  Level by witch the enemy is slowed.

- bool **at_end_of_path_**

  True if the enemy has reached the end of the path.

7.5.1 Detailed Description

Enemy class represents the blueprint of a basic enemy in the game. The base enemy class can be derived further to create towers with special powers.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Enemy()

```
td::Enemy::Enemy (
            types::Position position,
            float hitbox,
            sf::Texture * texture,
            float health = 100.0f,
            int move_speed = 1,
            int bounty = 0,
            bool armored = false,
            float distance_moved = 0.0f,
            unsigned int slowed_level = 0 )
```

Enemy constructor.

Parameters

position	Position of the enemy
hitbox	Radius of the circular region occupied by the enemy
texture	A pointer to the texture of the enemy
health	Health of the enemy
move_speed	Movement speed of the enemy
bounty	Bounty of the enemy
armored	Status ofenemy armor
moved_distance	Total distance moved on the path by the enemy
slowed_level	The level by which the enemy is slowed
melting_level	The level by which the enemy is melted

7.5.3 Member Function Documentation

7.5.3.1 createBasicCockroach()

```
Enemy td::Enemy::createBasicCockroach (
            types::Position startOfTheMap,
            sf::Texture * texture )
```

Create basic cockroach at the start of the map.

Parameters

startOfTheMap	Position of the start of the map
texture	Texture of the enemy

Returns

The basic cockroach

7.5.3.2 createBeetle()

```
Enemy td::Enemy::createBeetle (
            types::Position startOfTheMap,
            sf::Texture * texture )
```

Create beetle at the start of the map.

Parameters

startOfTheMap	Position of the start of the map
texture	Texture of the enemy

Returns

The beetle

7.5.3.3 createDragonfly()

```
Enemy td::Enemy::createDragonfly (
            types::Position startOfTheMap,
            sf::Texture * texture )
```

Create dragonfly (boss enemy) at the start of the map.

Parameters

startOfTheMap	Position of the start of the map
texture	Texture of the enemy

Returns

The dragonfly (boss)

7.5.3.4 createFly()

```
Enemy td::Enemy::createFly (
            types::Position startOfTheMap,
            sf::Texture * texture )
```

Create fly at the start of the map.

Parameters

startOfTheMap	Position of the start of the map
texture	Texture of the enemy

Returns

The fly

7.5.3.5 getBounty()

```
int td::Enemy::getBounty ( ) const
```

Get the bounty of the enemy.

Returns

Bounty of the enemy

7.5.3.6 getDistanceMoved()

```
float td::Enemy::getDistanceMoved ( ) const
```

Get the distance enemy has travelled.

Returns

Distance the enemy has travelled

7.5.3.7 getHealth()

```
float td::Enemy::getHealth ( ) const
```

Get the remaining health of the enemy.

Returns

Remaining health of the enemy

7.5.3.8 getMaxHealth()

```
float td::Enemy::getMaxHealth ( ) const
```

Get the maximum health of the enemy.

Returns

Maximum health of the enemy

7.5.3.9 getMoveSpeed()

```
int td::Enemy::getMoveSpeed ( ) const
```

Get the movement speed of the enemy.

Returns

Movement speed of the enemy

7.5.3.10 getSlowedLevel()

```
unsigned int td::Enemy::getSlowedLevel ( ) const
```

Get the slowed level of the enemy.

Returns

slowed_level of the enemy

7.5.3.11 isArmored()

```
bool td::Enemy::isArmored ( ) const
```

Status of enemy armor.

Returns

True if the enemy is armored otherwise false

7.5.3.12 isAtEndOfPath()

```
bool td::Enemy::isAtEndOfPath ( ) const
```

Returns

True if the enemy is at the end of the path

7.5.3.13 setHealth()

```
void td::Enemy::setHealth (
             float health_decrease )
```

Set the health of the enemy.

Parameters

health_decrease	the amount of health that is decreased from enemys health

7.5.3.14 TakeDamage()

```
bool td::Enemy::TakeDamage (
             float damage,
             bool is_armor_piercing )
```

called when a tower or projectile wants to do damage to the enemy

Parameters

damage	how much damage is dealt
is_armor_piercing	whether the damage dealt pierces armor

Returns

true if the enemy took damage

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

- tower-defense-5/include/enemy.hpp
- tower-defense-5/src/logic/enemies/enemy.cpp

7.6 td::Game Class Reference

Classes

- struct Wave

  A struct for storing the state of an enemy wave. A round consists of these waves.

Public Member Functions

- Game (Map ∗map, const std::string &round_file_path, const std::map< std::string, sf::Texture ∗ > &textures)

  Constructs a game with 2000 money and 100 lives.
- Game (Map ∗map, const std::string &round_file_path, int starting_money, int starting_lives, const std::map< std::string, sf::Texture ∗ > &textures)

  Constructs a game.
- int getMoney () const
- int getLives () const
- void **Update** ()
- std::list< Enemy > & getEnemies ()
- const std::list< Enemy > & getEnemies () const
- std::list< Tower > & getTowers ()
- const std::list< Tower > & getTowers () const
- std::list< Projectile > & getProjectiles ()
- const std::list< Projectile > & getProjectiles () const
- bool getAutoStart () const
- void setAutoStart (bool auto_start)
- bool SpawnEnemy (const std::string &enemy_identifier, types::Position position)

  Spawn an enemy on the map at the first path vertex.
- bool AddEnemy (const std::string &enemy_identifier, Enemy enemy)

  Add a possible enemy to the game.
- void AddTower (td::Tower &tower)

  Add a tower onto the map.
- const std::map< const Enemy ∗, std::vector< const Projectile ∗ > > & getEnemyCollisions (bool previous←_update=false)
- const std::map< const Projectile ∗, std::vector< const Enemy ∗ > > & getProjectileCollisions (bool previous_update=false)
- const Map ∗ getMap () const
- Map ∗ getMap ()
- void UpgradeTower (Tower ∗tower)

  Upgrades the tower given as the parameter if the player has enough money.
- void SellTower (Tower ∗tower)

  Sells the tower given as a parameter, deleting it and adding money to the player's balance.

- std::optional< Tower > StartBuyingTower (std::string name, sf::Texture ∗tower_texture, sf::Texture ∗projectile_texture, sf::Texture ∗extra_texture=nullptr)

  Begins the buying process by returning the appropriate tower to application if the player has enough money.
- const std::vector< std::vector< Wave > > & getRounds ()
- void LoadRounds (const std::string &file_path)

  Adds rounds from a JSON file. Doesn't remove pre-existing rounds.
- bool CheckTowerPlacementCollision (const Tower &tower)

  Check for collisions with blocked regions, existing towers and window walls when placing a tower.
- void StartRound (size_t round_index)

  Gives Game permission to start spawning in the enemies of a given round.
- bool IsRoundInProgress ()
- size_t getCurrentRoundIndex ()

  Round index increments when a new round starts.
- size_t getMaxRoundIndex ()
- void **Unpause** ()

  Resets the clock that calculates dt.

7.6.1 Constructor & Destructor Documentation

7.6.1.1 Game() [1/2]

```
td::Game::Game (
            Map * map,
            const std::string & round_file_path,
            const std::map< std::string, sf::Texture * > & textures )
```

Constructs a game with 2000 money and 100 lives.

Parameters

map	A pointer to the Map the game is played on.
round_file_path	File path to a JSON file that defines the game rounds
textures	The texture map provided by Application

7.6.1.2 Game() [2/2]

```
td::Game::Game (
            Map * map,
            const std::string & round_file_path,
            int starting_money,
            int starting_lives,
            const std::map< std::string, sf::Texture * > & textures )
```

Constructs a game.

Parameters

map	A pointer to the Map the game is played on.
starting_money	The amount of money the player has at the start
starting_lives	The amount of lives the player has at the start
textures	The texture map provided by Application

7.6.2 Member Function Documentation

7.6.2.1 AddEnemy()

```
bool td::Game::AddEnemy (
             const std::string & enemy_identifier,
             Enemy enemy )
```

Add a possible enemy to the game.

If an enemy with the identifier already exists, the method fails silently.

Parameters

enemy_identifier	A unique identifier for the enemy
enemy	An enemy instance to add to the game

Returns

True if the enemy was added, false otherwise

7.6.2.2 AddTower()

```
void td::Game::AddTower (
             td::Tower & tower )
```

Add a tower onto the map.

Parameters

tower	The tower to add

7.6.2.3 CheckTowerPlacementCollision()

```
bool td::Game::CheckTowerPlacementCollision (
             const Tower & tower )
```

Check for collisions with blocked regions, existing towers and window walls when placing a tower.

Parameters

tower	Tower that is being bought to be checked for collisions

Returns

True if there is any collision, false otherwise

7.6.2.4 getAutoStart()

```
bool td::Game::getAutoStart ( ) const
```

Returns

Whether auto starting rounds is on or not

7.6.2.5 getCurrentRoundIndex()

```
size_t td::Game::getCurrentRoundIndex ( )
```

Round index increments when a new round starts.

Returns

The one-indexed index of the current round

7.6.2.6 getEnemies() `[1/2]`

```
std::list< Enemy > & td::Game::getEnemies ( )
```

Returns

All the enemies currently on the map

7.6.2.7 getEnemies() [2/2]

```
const std::list< Enemy > & td::Game::getEnemies ( ) const
```

Returns

All the enemies currently on the (const)

7.6.2.8 getEnemyCollisions()

```
const std::map< const Enemy *, std::vector< const Projectile * > > & td::Game::getEnemy↩
Collisions (
             bool previous_update = false )
```

Parameters

previous_update	If set to true, returns collisions from the previous update

Returns

A map mapping enemies to projectiles that they collide with

7.6.2.9 getLives()

```
int td::Game::getLives ( ) const
```

Returns

Amount of lives left

7.6.2.10 getMap() [1/2]

```
Map * td::Game::getMap ( )
```

Returns

A pointer to the map

7.6.2.11 getMap() [2/2]

```
const Map * td::Game::getMap ( ) const
```

Returns

A const pointer to the map

7.6.2.12 getMaxRoundIndex()

```
size_t td::Game::getMaxRoundIndex ( )
```

Returns

The index of the final round

7.6.2.13 getMoney()

```
int td::Game::getMoney ( ) const
```

Returns

Amount of money the player has

7.6.2.14 getProjectileCollisions()

```
const std::map< const Projectile *, std::vector< const Enemy * > > & td::Game::getProjectile←
Collisions (
             bool previous_update = false )
```

Parameters

previous_update	If set to true, returns collisions from the previous update

Returns

A map mapping projectiles enemies that they collide with

7.6.2.15 getProjectiles() [1/2]

```
std::list< Projectile > & td::Game::getProjectiles ( )
```

Returns

All the projectiles currently on the map

7.6.2.16 getProjectiles() [2/2]

```
const std::list< Projectile > & td::Game::getProjectiles ( ) const
```

Returns

All the projectiles currently on the map (const)

7.6.2.17 getRounds()

```
const std::vector< std::vector< Game::Wave > > & td::Game::getRounds ( )
```

Returns

A vector of rounds, with each round being a vector consisting of Game::Wave elements (waves)

7.6.2.18 getTowers() [1/2]

```
std::list< Tower > & td::Game::getTowers ( )
```

Returns

All the towers currently on the map

7.6.2.19 getTowers() [2/2]

```
const std::list< Tower > & td::Game::getTowers ( ) const
```

Returns

All the towers currently on the map (const)

7.6.2.20 IsRoundInProgress()

```
bool td::Game::IsRoundInProgress ( )
```

Returns

True if a round is in progress, false otherwise

7.6.2.21 LoadRounds()

```
void td::Game::LoadRounds (
             const std::string & file_path )
```

Adds rounds from a JSON file. Doesn't remove pre-existing rounds.

The JSON file consists of an array of arrays that all contain an object of the form { "enemyIdentifier": "asd", "spacing": 500, "offset": 0, "count": 5}

7.6.2.22 SellTower()

```
void td::Game::SellTower (
             Tower * tower )
```

Sells the tower given as a parameter, deleting it and adding money to the player's balance.

Parameters

tower	The tower being sold

7.6.2.23 setAutoStart()

```
void td::Game::setAutoStart (
             bool auto_start )
```

Parameters

auto_start	Whether auto starting rounds is set on or off

7.6.2.24 SpawnEnemy()

```
bool td::Game::SpawnEnemy (
            const std::string & enemy_identifier,
            types::Position position )
```

Spawn an enemy on the map at the first path vertex.

The method fails silently if the specified identifier is invalid.

Parameters

enemy_identifier	A unique identifier for the enemy, has to be added to the game first using the AddEnemy() method.
position	The enemy's initial position.

Returns

True if the enemy was spawned successfully, false otherwise.

7.6.2.25 StartBuyingTower()

```
std::optional< Tower > td::Game::StartBuyingTower (
            std::string name,
            sf::Texture * tower_texture,
            sf::Texture * projectile_texture,
            sf::Texture * extra_texture = nullptr )
```

Begins the buying process by returning the appropriate tower to application if the player has enough money.

Parameters

name	Identifier used to map to a tower object
tower_texture	Pointer to the texture of the tower
projectile_texture	Pointer to the texture of the projectile
extra_texture	Pointer to another texture a tower might use (like bomb explosion)

7.6.2.26 StartRound()

```
void td::Game::StartRound (
            size_t round_index )
```

Gives Game permission to start spawning in the enemies of a given round.

Parameters

round_index	A zero-indexed index

7.6.2.27 UpgradeTower()

```
void td::Game::UpgradeTower (
            Tower * tower )
```

Upgrades the tower given as the parameter if the player has enough money.

Parameters

tower	The tower being upgraded

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

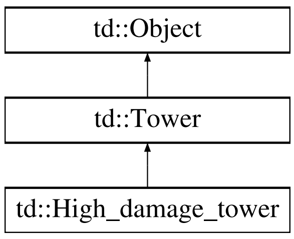
- tower-defense-5/include/game.hpp
- tower-defense-5/src/logic/game.cpp

7.7 td::High_damage_tower Class Reference

//Implementation of high damage tower that has sparse shooting speed, but does massive damage on one hit.

```
#include <high_damage_tower.hpp>
```

Inheritance diagram for td::High_damage_tower:



Public Member Functions

- High_damage_tower (types::Position position, float rotation_angle=0.0f, sf::Texture ∗texture=nullptr, sf::↩
Texture ∗texture_projectile=nullptr)

    High_damage_tower constructor.
- void Upgrade ()

    Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

    Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &, std::list< Enemy > &enemies)

    Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.7.1 Detailed Description

//Implementation of high damage tower that has sparse shooting speed, but does massive damage on one hit.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 High_damage_tower()

```
td::High_damage_tower::High_damage_tower (
             types::Position position,
             float rotation_angle = 0.0f,
             sf::Texture * texture = nullptr,
             sf::Texture * texture_projectile = nullptr )
```

High_damage_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.7.3 Member Function Documentation

7.7.3.1 Shoot()

```
bool td::High_damage_tower::Shoot (
             std::list< Projectile > & projectiles,
             std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

Reimplemented from td::Tower.

7.7.3.2 Update()

```
void td::High_damage_tower::Update (
            types::Time dt,
            std::list< Enemy > & enemies,
            std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.7.3.3 Upgrade()

```
void td::High_damage_tower::Upgrade ( )  [virtual]
```

Upgrades the tower once.

Reimplemented from td::Tower.

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

- tower-defense-5/include/high_damage_tower.hpp
- tower-defense-5/src/logic/towers/high_damage_tower.cpp

7.8 td::Map Class Reference

Public Member Functions

- Map (const std::string &background_image_path, std::vector< td::types::Position > enemy_path, std↩::vector< td::types::BlockedRegion > blocked_regions)
- const std::string & getBackgroundImagePath ()
- std::vector< td::types::Position > & getEnemyPath ()
- const std::vector< td::types::Position > & getEnemyPath () const
- std::vector< td::types::BlockedRegion > & getBlockedRegions ()
- const std::vector< td::types::BlockedRegion > & getBlockedRegions () const
- td::types::Position GetStartingPosition ()

Static Public Member Functions

- static Map ∗ LoadFromFile (const std::string &file_name)

7.8.1 Constructor & Destructor Documentation

7.8.1.1 Map()

```
td::Map::Map (
             const std::string & background_image_path,
             std::vector< td::types::Position > enemy_path,
             std::vector< td::types::BlockedRegion > blocked_regions )
```

Parameters

background_image_path	Path to the map background
enemy_path	Defines the path that the enemies take
blocked_regions	Defines regions that towers can't be placed on

7.8.2 Member Function Documentation

7.8.2.1 getBackgroundImagePath()

`const std::string & td::Map::getBackgroundImagePath ( )`

Returns

A path to the background image

7.8.2.2 getBlockedRegions() `[1/2]`

`std::vector< td::types::BlockedRegion > & td::Map::getBlockedRegions ( )`

Returns

A reference to a vector of the blocked regions

7.8.2.3 getBlockedRegions() `[2/2]`

`const std::vector< td::types::BlockedRegion > & td::Map::getBlockedRegions ( ) const`

Returns

A const reference to a vector of the blocked regions

7.8.2.4 getEnemyPath() `[1/2]`

`std::vector< td::types::Position > & td::Map::getEnemyPath ( )`

Returns

A reference to the enemy path

7.8.2.5 getEnemyPath() [2/2]

```
const std::vector< td::types::Position > & td::Map::getEnemyPath ( ) const
```

Returns

A const reference to the enemy path

7.8.2.6 GetStartingPosition()

```
td::types::Position td::Map::GetStartingPosition ( )
```

Returns

The position where enemies are supposed to spawn

7.8.2.7 LoadFromFile()

```
Map * td::Map::LoadFromFile (
             const std::string & file_name )  [static]
```

Parameters

file_name	Path to the JSON file to load the Map from

Returns

A pointer to a Map that was loaded from the given file

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

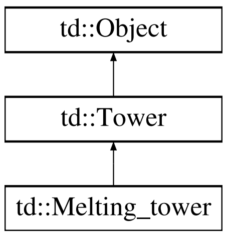
- tower-defense-5/include/map.hpp
- tower-defense-5/src/logic/map.cpp

7.9 td::Melting_tower Class Reference

//Implementation of melting tower that damages every enemy in its range

```
#include <melting_tower.hpp>
```

Inheritance diagram for td::Melting_tower:



Public Member Functions

- Melting_tower (sf::Vector2< float > position, float rotation_angle=0.0f, sf::Texture ∗texture=nullptr)

  Melting_tower constructor.
- void Upgrade ()

  Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

  Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &, std::list< Enemy > &enemies)

  Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.9.1 Detailed Description

//Implementation of melting tower that damages every enemy in its range

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Melting_tower()

```
td::Melting_tower::Melting_tower (
             sf::Vector2< float > position,
             float rotation_angle = 0.0f,
             sf::Texture ∗ texture = nullptr )
```

Melting_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.9.3 Member Function Documentation

7.9.3.1 Shoot()

```
bool td::Melting_tower::Shoot (
            std::list< Projectile > & projectiles,
            std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

Reimplemented from td::Tower.

7.9.3.2 Update()

```
void td::Melting_tower::Update (
            types::Time dt,
            std::list< Enemy > & enemies,
            std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.9.3.3 Upgrade()

```
void td::Melting_tower::Upgrade ( )  [virtual]
```

Upgrades the tower once.

Reimplemented from td::Tower.

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

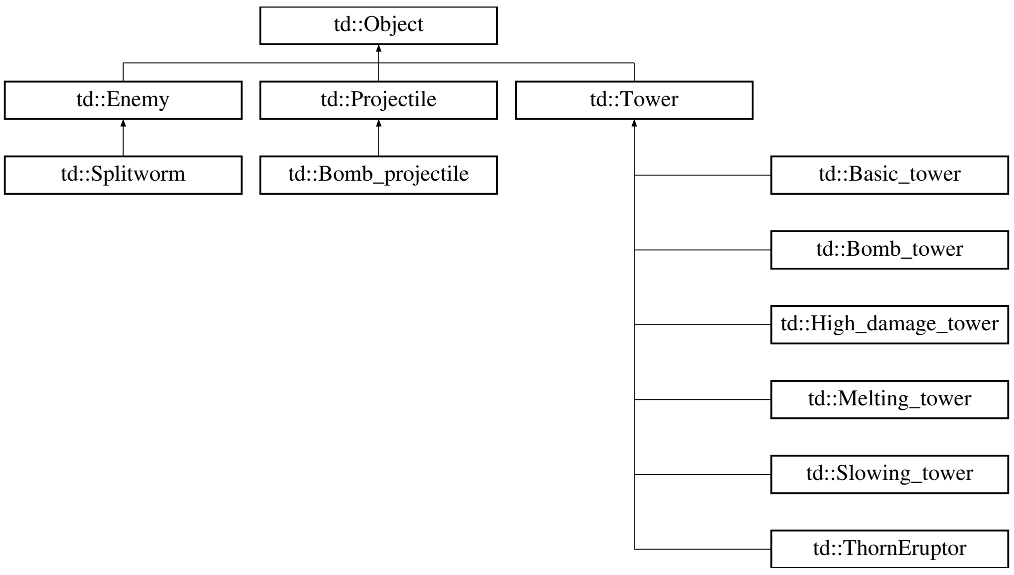
- tower-defense-5/include/melting_tower.hpp
- tower-defense-5/src/logic/towers/melting_tower.cpp

7.10 td::Object Class Reference

Object class is an abstract class representing the basic entity of the game. The object class shall be inherited by all other game entities such as towers, enemies, etc.

```
#include <object.hpp>
```

Inheritance diagram for td::Object:



Public Member Functions

- Object (types::Position position=types::Position(0.0f, 0.0f), float hitboxRadius=0.0f, sf::Texture ∗texture=nullptr, float rotation_angle=0.0f)

  Object constructor.
- **Object** (const Object &obj)
- virtual types::Position getPosition () const

  Get the position of the object.
- virtual float getHitboxRadius () const

  Get the radius of circular region occupied by the object.
- virtual const sf::Texture ∗ getTexture () const

  Get a const pointer to the texture of the object.
- virtual sf::Texture ∗ getTexture ()

  Get a pointer to the texture of the object.
- virtual void setPosition (types::Position position)

  Set the position of the object.
- virtual void setRotation (float angle)

  Set the rotation of the object.
- virtual float getRotation () const

  Get the orientation of the object.
- void **Delete** ()

  Signal Game that this Object should not make it to the next game update.
- bool IsDeleted ()

Protected Attributes

- types::Position **position_**

  Position of the object.
- float **hitboxRadius_**

  Radius of the circular region occupied by the object.
- sf::Texture ∗ **texture_**

  Texture of the object.
- float **rotation_angle_**

  Orientation, in radians of the object.

7.10.1 Detailed Description

Object class is an abstract class representing the basic entity of the game. The object class shall be inherited by all other game entities such as towers, enemies, etc.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Object()

```
td::Object::Object (
            types::Position position = types::Position(0.0f, 0.0f),
            float hitboxRadius = 0.0f,
            sf::Texture * texture = nullptr,
            float rotation_angle = 0.0f )
```

Object constructor.

Parameters

position	Position of the object
hitboxRadius	Radius of the circular region occupied by the object
texture	Texture of the object
rotation_angle	Orientation, in radians of the object

7.10.3 Member Function Documentation

7.10.3.1 getHitboxRadius()

```
float td::Object::getHitboxRadius ( ) const  [virtual]
```

Get the radius of circular region occupied by the object.

Returns

Radius of the circular region occupied by the object

7.10.3.2 getPosition()

`types::Position td::Object::getPosition ( ) const [virtual]`

Get the position of the object.

Returns

Position of the object

7.10.3.3 getRotation()

`float td::Object::getRotation ( ) const [virtual]`

Get the orientation of the object.

Returns

Orientation, in radians of the object

7.10.3.4 getTexture() [1/2]

`sf::Texture * td::Object::getTexture ( ) [virtual]`

Get a pointer to the texture of the object.

Returns

Texture of the object

7.10.3.5 getTexture() [2/2]

`const sf::Texture * td::Object::getTexture ( ) const [virtual]`

Get a const pointer to the texture of the object.

Returns

Texture of the object

7.10.3.6 IsDeleted()

```
bool td::Object::IsDeleted ( )
```

Returns

True if the Object should not make it to the next game update, false otherwise

7.10.3.7 setPosition()

```
void td::Object::setPosition (
            types::Position position )  [virtual]
```

Set the position of the object.

Parameters

position	Position of the object

7.10.3.8 setRotation()

```
void td::Object::setRotation (
            float angle )  [virtual]
```

Set the rotation of the object.

Parameters

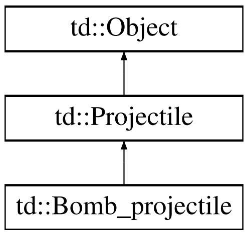
angle	Orientation of the object

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

- tower-defense-5/include/object.hpp
- tower-defense-5/src/logic/object.cpp

7.11 td::Projectile Class Reference

Inheritance diagram for td::Projectile:



Public Member Functions

- Projectile (types::Position position, float hitbox, sf::Texture ∗texture, float rotation_angle, float damage, bool is_armor_piercing, unsigned int piercing, float speed, float lifetime)

    Projectile constructor.
- virtual void **Update** (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)
- void **Update** (types::Time dt)
- float getDamage () const

    Get the damage value of the projectile.
- bool isArmorPiercing () const

    Get the armor piercing status of the projectile.
- void setPiercingLeft (unsigned int count)

    Set the number of enemies pierced by the projectile.
- float getSpeed () const

    Get the speed value of the projectile.
- float getLifetimeLeft () const

    Get the lifetime the projectile has left.
- unsigned int getPiercingLeft () const

    Get the number of enemies pierced by the projectile.

Protected Attributes

- float **damage_**

    Damage value of the projectile.
- bool **is_armor_piercing_**

    Armor piercing status of the projectile.
- unsigned int piercing_left_

    projectile before disappearing
- float **speed_**

    Speed of the projectile.
- float **lifetime_left_**

    How much distance the projectile can still travel before disappearing.

7.11.1 Constructor & Destructor Documentation

7.11.1.1 Projectile()

```
td::Projectile::Projectile (
            types::Position position,
            float hitbox,
            sf::Texture * texture,
            float rotation_angle,
            float damage,
            bool is_armor_piercing,
            unsigned int piercing,
            float speed,
            float lifetime )
```

Projectile constructor.

Parameters

position	Position of the projectile
hitbox	Radius of the circular region occupied by the projectile
texture	Texture of the projectile
rotation_angle	Orientation, in radians of the projectile
damage	Damage % of the projectile
is_armor_piercing	Status of projectile armor
piercing	Number of enemies pierced by the projectile before disappearing

7.11.2 Member Function Documentation

7.11.2.1 getDamage()

```
float td::Projectile::getDamage ( ) const
```

Get the damage value of the projectile.

Returns

Damage value of the projectile

7.11.2.2 getLifetimeLeft()

```
float td::Projectile::getLifetimeLeft ( ) const
```

Get the lifetime the projectile has left.

Returns

How far the projectile can still travel

7.11.2.3 getPiercingLeft()

```
unsigned int td::Projectile::getPiercingLeft ( ) const
```

Get the number of enemies pierced by the projectile.

Returns

Number of enemies pierced by the projectile

7.11.2.4 getSpeed()

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

Get the speed value of the projectile.

Returns

Speed value of the projectile

7.11.2.5 isArmorPiercing()

```
bool td::Projectile::isArmorPiercing ( ) const
```

Get the armor piercing status of the projectile.

Returns

True if projectile has armor piercing active, false otherwise

7.11.2.6 setPiercingLeft()

```
void td::Projectile::setPiercingLeft (
            unsigned int count )
```

Set the number of enemies pierced by the projectile.

Parameters

count	Number of enemies pierced by the projectile

7.11.3 Member Data Documentation

7.11.3.1 piercing_left_

```
unsigned int td::Projectile::piercing_left_  [protected]
```

projectile before disappearing

Number of enemies pierced by the

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

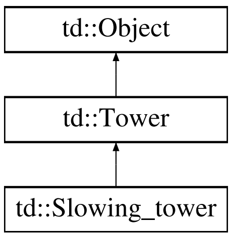
- tower-defense-5/include/projectile.hpp
- tower-defense-5/src/logic/projectiles/projectile.cpp

7.12 td::Slowing_tower Class Reference

//Implementation of the slowing tower that slows down enemies in its range

```
#include <slowing_tower.hpp>
```

Inheritance diagram for td::Slowing_tower:



Public Member Functions

- Slowing_tower (sf::Vector2< float > position, float rotation_angle=0.0f, sf::Texture ∗texture=nullptr)

  Slowing_tower constructor.
- void Upgrade ()

  Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

  Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &, std::list< Enemy > &enemies)

  Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.12.1 Detailed Description

//Implementation of the slowing tower that slows down enemies in its range

7.12.2 Constructor & Destructor Documentation

7.12.2.1 Slowing_tower()

```
td::Slowing_tower::Slowing_tower (
             sf::Vector2< float > position,
             float rotation_angle = 0.0f,
             sf::Texture * texture = nullptr )
```

Slowing_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.12.3 Member Function Documentation

7.12.3.1 Shoot()

```
bool td::Slowing_tower::Shoot (
             std::list< Projectile > & projectiles,
             std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

NOTE! Before starting each frame or time segment the slowed_level_ parameters of all enemies should be set to 0!

if enemy is not in range of another slowing tower that has bigger level, set slowing_level to be same as level of this tower. it->getBounty() != 400 makes dragonfly immune to slowring tower

Reimplemented from td::Tower.

7.12.3.2 Update()

```
void td::Slowing_tower::Update (
             types::Time dt,
             std::list< Enemy > & enemies,
             std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.12.3.3 Upgrade()

```
void td::Slowing_tower::Upgrade ( )  [virtual]
```

Upgrades the tower once.

Reimplemented from td::Tower.

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

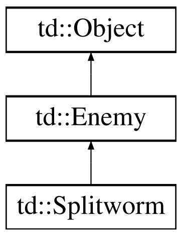
- tower-defense-5/include/slowing_tower.hpp
- tower-defense-5/src/logic/towers/slowing_tower.cpp

7.13 td::Splitworm Class Reference

//Implementation of splitworm

```
#include <splitworm.hpp>
```

Inheritance diagram for td::Splitworm:



Public Member Functions

- Splitworm (types::Position startOfTheMap, float hitbox, sf::Texture ∗texture, float health=150, int move_speed=20, float bounty=0.0f, bool armored=true, float distance_moved=0.0f)

    Splitworm constructor.
- std::vector< Enemy > doUponDeath (sf::Texture ∗texture)

    Splitworm creates 3 weaker worm upon death.

Additional Inherited Members

7.13.1 Detailed Description

//Implementation of splitworm

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Splitworm()

```
td::Splitworm::Splitworm (
            types::Position startOfTheMap,
            float hitbox,
            sf::Texture * texture,
            float health = 150,
            int move_speed = 20,
            float bounty = 0.0f,
            bool armored = true,
            float distance_moved = 0.0f )
```

Splitworm constructor.

Parameters

position	Position of the enemy
hitbox	Radius of the circular region occupied by the enemy
texture	A pointer to the texture of the enemy
health	Health of the enemy
move_speed	Movement speed of the enemy
bounty	Bounty of the enemy
armored	Status ofenemy armor
moved_distance	Total distance moved on the path by the enemy

7.13.3 Member Function Documentation

7.13.3.1 doUponDeath()

```
std::vector< Enemy > td::Splitworm::doUponDeath (
            sf::Texture * texture )
```

Splitworm creates 3 weaker worm upon death.

Parameters

texture	Texture of the weaker worm

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

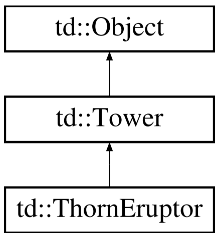
- tower-defense-5/include/splitworm.hpp
- tower-defense-5/src/logic/enemies/splitworm.cpp

7.14 td::ThornEruptor Class Reference

//Implementation of our most basic tower type

```
#include <thorn_eruptor.hpp>
```

Inheritance diagram for td::ThornEruptor:



Public Member Functions

- ThornEruptor (types::Position position, float rotation_angle=0.0f, sf::Texture ∗texture=nullptr, sf::Texture ∗texture_projectile=nullptr)

    Basic_tower constructor.
- void Upgrade ()

    Upgrades the tower once.
- void Update (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)

    Run every frame, handles the tower's functionality.
- bool Shoot (std::list< Projectile > &projectiles, std::list< Enemy > &enemies)

    Add projectiles shoot by the tower to the list of all projectiles.

Additional Inherited Members

7.14.1 Detailed Description

//Implementation of our most basic tower type

7.14.2 Constructor & Destructor Documentation

7.14.2.1 ThornEruptor()

```
td::ThornEruptor::ThornEruptor (
             types::Position position,
             float rotation_angle = 0.0f,
             sf::Texture * texture = nullptr,
             sf::Texture * texture_projectile = nullptr )
```

Basic_tower constructor.

Parameters

position	Position of the tower
rotation_angle	Orientation, in radians of the tower

7.14.3 Member Function Documentation

7.14.3.1 Shoot()

```
bool td::ThornEruptor::Shoot (
             std::list< Projectile > & projectiles,
             std::list< Enemy > & enemies )  [virtual]
```

Add projectiles shoot by the tower to the list of all projectiles.

Returns

List of all projectiles currently in game

Reimplemented from td::Tower.

7.14.3.2 Update()

```
void td::ThornEruptor::Update (
             types::Time dt,
             std::list< Enemy > & enemies,
             std::list< Projectile > & projectiles )
```

Run every frame, handles the tower's functionality.

Parameters

dt	time since last call to this function
enemies	enemies in the game
projectiles	projectiles in the game

7.14.3.3 Upgrade()

```
void td::ThornEruptor::Upgrade ( )  [virtual]
```

Upgrades the tower once.

Reimplemented from td::Tower.

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

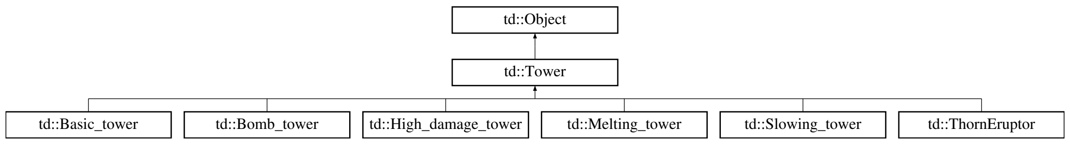
- tower-defense-5/include/thorn_eruptor.hpp
- tower-defense-5/src/logic/towers/thorn_eruptor.cpp

7.15 td::Tower Class Reference

Tower class represents the blueprint of a basic tower in the game. The base tower class can be derived further to create towers with special powers.

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

Inheritance diagram for td::Tower:



Public Member Functions

- Tower (types::Position position, float hitbox, sf::Texture ∗texture, sf::Texture ∗texture_projectile, float rotation←_angle=0.0f, unsigned int attack_speed=1U, float range=1.0f, unsigned int cost=0, unsigned int upgrade_←cost=0, unsigned int level=1, types::Targeting targetTo=types::kFirst)

  Tower constructor.
- void **Update** (types::Time dt, const td::Game &)
- void **Update** (types::Time dt, std::list< Enemy > &enemies, std::list< Projectile > &projectiles)
- Tower (types::Position position, float rotation_angle, unsigned int attack_speed)

  Tower constructor.
- unsigned int getAttackSpeed () const

  Get the attack speed of the tower.
- float getRange () const

  Get the attack range of the tower.
- unsigned int getLevel () const

  Get the level of the tower.
- unsigned int getCost () const

  Get the price of the tower.
- unsigned int getMoneySpent () const

Get the total amount of money spent on the tower.

- void setMoneySpent (unsigned int value)

  Set the total amount of money spent on the tower.
- unsigned int getUpgradeCost () const

  Get the upgrade cost of the tower at current level.
- const std::string & getName () const

  Get the name of the tower.
- types::Targeting getTargeting () const

  Get the target type of the tower.
- void setTargeting (types::Targeting targeting)

  Set the target type of the tower.
- virtual bool Shoot (std::list< Projectile > &projectiles, std::list< Enemy > &enemies)

  Get the shooting type of the tower.
- virtual void Upgrade ()

  Upgrades the tower once.
- virtual std::optional< Enemy ∗ > GetTarget (std::list< Enemy > &enemies)

  Get the enemy tower is targeting.
- types::Position GetProjectStartPos ()

  Calculate the starting position of the projectiles shoot by the tower.

Protected Attributes

- std::string **name_**

  Name that is used as an identifier.
- unsigned int **attack_speed_**

  Attack speed of the tower.
- float **range_**

  Attack range of the tower.
- unsigned int **level_**

  Level of the tower.
- types::Targeting **targeting_**

  Target mode of the tower.
- sf::Texture ∗ texture_projectile_
- unsigned int **cost_**

  cost of the tower
- unsigned int **upgrade_cost_**

  upgrade cost of the tower
- unsigned int money_spent_on_tower_
- types::Time **time_since_last_shoot_**

7.15.1 Detailed Description

Tower class represents the blueprint of a basic tower in the game. The base tower class can be derived further to create towers with special powers.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 Tower() [1/2]

```
td::Tower::Tower (
            types::Position position,
            float hitbox,
            sf::Texture * texture,
            sf::Texture * texture_projectile,
            float rotation_angle = 0.0f,
            unsigned int attack_speed = 1U,
            float range = 1.0f,
            unsigned int cost = 0,
            unsigned int upgrade_cost = 0,
            unsigned int level = 1,
            types::Targeting targetTo = types::kFirst )
```

Tower constructor.

Parameters

position	Position of the tower
hitbox	Radius of the circular region occupied by the tower
texture	Texture of the tower
attack_speed	Attack speed of the tower
range	Attack range of the tower
level	Level of the tower
targetTo	Stores the target mode of the tower, options: c = closest, s = strongest or f = furthest travelled

7.15.2.2 Tower() [2/2]

```
td::Tower::Tower (
            types::Position position,
            float rotation_angle,
            unsigned int attack_speed )  [explicit]
```

Tower constructor.

Parameters

position	Position of the tower
attack_speed	Attack speed of the tower

7.15.3 Member Function Documentation

7.15.3.1 getAttackSpeed()

```
unsigned int td::Tower::getAttackSpeed ( ) const
```

Get the attack speed of the tower.

Returns

Attack speed of the tower

7.15.3.2 getCost()

```
unsigned int td::Tower::getCost ( ) const
```

Get the price of the tower.

Returns

Cost of the tower

7.15.3.3 getLevel()

```
unsigned int td::Tower::getLevel ( ) const
```

Get the level of the tower.

Returns

Level of the tower

7.15.3.4 getMoneySpent()

```
unsigned int td::Tower::getMoneySpent ( ) const
```

Get the total amount of money spent on the tower.

Returns

The total amount of money spent on the tower

7.15.3.5 getName()

```
const std::string & td::Tower::getName ( ) const
```

Get the name of the tower.

Returns

Name of the tower

7.15.3.6 GetProjectStartPos()

```
types::Position td::Tower::GetProjectStartPos ( )
```

Calculate the starting position of the projectiles shoot by the tower.

Parameters

centre	Centre position of the tower
radius	Radius of the tower (same as hitbox)
angle	Angle position of the tower in radians

7.15.3.7 getRange()

```
float td::Tower::getRange ( ) const
```

Get the attack range of the tower.

Returns

Attack range of the tower

7.15.3.8 GetTarget()

```
std::optional< Enemy * > td::Tower::GetTarget (
            std::list< Enemy > & enemies )  [virtual]
```

Get the enemy tower is targeting.

Returns

Pointer to the targeted enemy

Parameters

enemies	vector of the enemies in current game

7.15.3.9 getTargeting()

```
types::Targeting td::Tower::getTargeting ( ) const
```

Get the target type of the tower.

Returns

Target type of the tower

7.15.3.10 getUpgradeCost()

```
unsigned int td::Tower::getUpgradeCost ( ) const
```

Get the upgrade cost of the tower at current level.

Returns

Upgrade cost of the tower

7.15.3.11 setMoneySpent()

```
void td::Tower::setMoneySpent (
             unsigned int value )
```

Set the total amount of money spent on the tower.

Parameters

value	new value for money_spent_on_tower_

7.15.3.12 setTargeting()

```
void td::Tower::setTargeting (
             types::Targeting targeting )
```

Set the target type of the tower.

Parameters

targetType	Target type of the tower

7.15.3.13 Shoot()

```
bool td::Tower::Shoot (
             std::list< Projectile > & projectiles,
             std::list< Enemy > & enemies ) [virtual]
```

Get the shooting type of the tower.

Returns

bool that tells if the tower shot

Parameters

projectiles	list of the projectiles in current game
enemies	list of the enemies in current game

Reimplemented in td::Basic_tower, td::Bomb_tower, td::High_damage_tower, td::Melting_tower, td::Slowing_tower, and td::ThornEruptor.

7.15.3.14 Upgrade()

```
virtual void td::Tower::Upgrade ( )  [inline], [virtual]
```

Upgrades the tower once.

Reimplemented in td::Basic_tower, td::Bomb_tower, td::High_damage_tower, td::Melting_tower, td::Slowing_tower, and td::ThornEruptor.

7.15.4 Member Data Documentation

7.15.4.1 money_spent_on_tower_

```
unsigned int td::Tower::money_spent_on_tower_  [protected]
```

Total money spent on this tower, used when selling tower

7.15.4.2 texture_projectile_

```
sf::Texture* td::Tower::texture_projectile_  [protected]
```

Pointer to texture of the projectile the tower shoots

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

- tower-defense-5/include/tower.hpp
- tower-defense-5/src/logic/towers/tower.cpp

7.16 td::Game::Wave Struct Reference

A struct for storing the state of an enemy wave. A round consists of these waves.

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

Public Member Functions

- Wave (std::string enemy_identifier, unsigned int spacing, unsigned int offset, unsigned int count)

Public Attributes

- std::string **enemy_identifier**
- unsigned int **spacing** = 500
- unsigned int **offset** = 0
- unsigned int **count** = 1
- unsigned int **enemies_spawned** = 0
- int **last_spawn_time** = 0

7.16.1 Detailed Description

A struct for storing the state of an enemy wave. A round consists of these waves.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 Wave()

```
td::Game::Wave::Wave (
            std::string enemy_identifier,
            unsigned int spacing,
            unsigned int offset,
            unsigned int count )  [inline]
```

Parameters

enemy_identifier	The unique identifier for the enemy that gets spawned during the wave
spacing	The amount of time between enemy spawns, in milliseconds
offset	The amount of time for the wave to arrive after the round has started, in milliseconds
count	The amount of enemies that spawn

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

- tower-defense-5/include/game.hpp

Chapter 8

File Documentation

8.1 application.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <TGUI/Backend/SFML-Graphics.hpp>
#include <TGUI/TGUI.hpp>
#include <map>
#include <optional>
#include <string>

#include "collision.hpp"
#include "enemy.hpp"
#include "game.hpp"
#include "object.hpp"
#include "projectile.hpp"
#include "tower.hpp"
#include "types.hpp"
#include "basic_tower.hpp"
#include "bomb_tower.hpp"
#include "high_damage_tower.hpp"
#include "melting_tower.hpp"
#include "slowing_tower.hpp"


namespace td {
class Application {
 public:
  Application();

  int run();

  void LoadTextures();

  void LaunchMainMenuGui();

  void LaunchMapSelectGui();

  void LaunchGameGui();

  void LaunchOptionsGui();

  void LaunchPauseGui();

  void LaunchUpgradeGui();

  void LaunchGame(const std::string& map_name);

  void CloseGame();

  void HandleMainMenu();

  void HandleMapSelect();

  void HandleGame();

  void HandleOptions();

  void HandlePause();

  void HandleUpgrade();

  void HandleMainMenuGui();

  void HandleMapSelectGui();

  void HandleGameGui();

  void HandleOptionsGui();

  void HandlePauseGui();

  void HandleUpgradeGui();

  void TargetingSwitchRight();

  void TargetingSwitchLeft();

  void DrawGameElements();

  void DrawShopElements();

  void ScaleSprite(sf::Transformable& sprite);

  void StyleButtonBrown(tgui::Button::Ptr button);

 private:
  sf::RenderWindow window_;
  unsigned int window_x_;   // width of the window on creation, could be global
                            // constant instead?
  unsigned int window_y_;   // height of the window on creation
  tgui::Gui gui_{window_};  // Gui object where widgets are added to
  types::AppState state_ =
      types::kOptions;  // tracks the state the application is in
  std::map<std::string, sf::Texture*> textures_;  // map with all loaded
                                                  // textures
  std::optional<Game> game_ = {};
  sf::Font font_;         // font used for sf::Text
  float volume_ = 70.0f;  // value between 0 and 100 that affects the volume of
                          // sound effects
  float music_volume_ = 70.0f;  // value between 0 and 100 that affects the
                                // volume of background music
  bool auto_start_ = false;
  Tower* upgrading_tower_ =
      nullptr;  // Tower that currently has its upgrade menu open,
                // nullptr if no tower is being upgraded
  std::optional<Tower> buying_tower_ =
      {};  // Tower that currently has its upgrade menu
           // open, nullptr if no tower is being upgraded

};
}  // namespace td
```

8.2 basic_tower.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"
#include "collision.hpp"


namespace td {
class Basic_tower : public Tower {
 public:
  Basic_tower(types::Position position, float rotation_angle = 0.0f, sf::Texture* texture = nullptr,
    sf::Texture* texture_projectile = nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);


  bool Shoot(std::list<Projectile>&, std::list<Enemy>& enemies);
};
}  // namespace td
```

8.3 bomb_projectile.hpp

```
#pragma once

#include <SFML/Graphics.hpp>

#include "projectile.hpp"

namespace td {
class Bomb_projectile : public Projectile {
 public:
  Bomb_projectile(types::Position position, float rotation_angle, int damage,
                  sf::Texture* texture_projectile, sf::Texture* texture_explosion,
                  float speed, float lifetime);

 private:
 sf::Texture* texture_explosion_;
};
}  // namespace td
```

8.4 bomb_tower.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"

namespace td {
class Bomb_tower : public Tower {
 public:
  Bomb_tower(sf::Vector2<float> position, float rotation_angle = 0.0f,
             sf::Texture* texture = nullptr,
             sf::Texture* texture_projectile = nullptr,
             sf::Texture* texture_explosion = nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  bool Shoot(std::list<Projectile>&, std::list<Enemy>& enemies);

 private:
  sf::Texture* texture_explosion_;
};
}  // namespace td
```

8.5 tower-defense-5/include/collision.hpp File Reference

This file contains functions to check collision between different shapes like collision between two circles, circle and a convex polygon. Call these functions wherever there is a need to check for collisions between the mentioned shapes.

```
#include <cmath>
#include "types.hpp"
```

Functions

- double td::EuclideanDistance (td::types::Position p1, td::types::Position p2)

  Function to calculate the distance between two points 1 and 2.
- bool td::IsPointBetween (td::types::Position a, td::types::Position p, td::types::Position b)

  Check if a given point p is between two points a and b.
- double td::Angle2D (double x1, double y1, double x2, double y2)

  Function to find the 2D angle between vectors (x1, y1) & (x2, y2)

- double td::DotProduct2D (td::types::Position a, td::types::Position b, td::types::Position c)

  Find the 2D dot product of 2 line segments made by points a, b & c.
- double td::CrossProduct2D (td::types::Position a, td::types::Position b, td::types::Position c)

  Find the 2D cross product of 2 line segments made by points a, b & c.
- bool td::IsCircleIntersectingLineSegment (td::types::Position p, float r, std::pair< td::types::Position, td::types::Position > line_segment)

  Check if a given circle intersects a given line segment or not.
- bool td::IsCircleCenterInsidePolygon (td::types::Position p, std::vector< std::pair< td::types::Position, td::types::Position > > edges)

  Function to check if the circle center is inside a polygon or not.
- bool td::IsCircleCollidingWithCircle (td::types::Position p1, float r1, td::types::Position p2, float r2)

  Function to check if a circle is colliding with another circle or not.
- bool td::IsCircleCollidingWithPolygon (td::types::Position p, float r, std::vector< td::types::Position > polygon_points)

  Function to check if a circle is colliding with a polygon or not.

Variables

- constexpr double **td::PI** = 3.14

  Constant for PI (in radians)

8.5.1 Detailed Description

This file contains functions to check collision between different shapes like collision between two circles, circle and a convex polygon. Call these functions wherever there is a need to check for collisions between the mentioned shapes.

8.5.2 Function Documentation

8.5.2.1 Angle2D()

```
double td::Angle2D (
            double x1,
            double y1,
            double x2,
            double y2 )
```

Function to find the 2D angle between vectors (x1, y1) & (x2, y2)

Parameters

x1	X coordinate of vector 1
y1	Y coordinate of vector 1
x2	X coordinate of vector 2
y2	Y coordinate of vector 2

Returns

Angle (in radians) made by vector 1 to vector 2. Angle is positive anticlockwise and between -PI to +PI.

8.5.2.2 CrossProduct2D()

```
double td::CrossProduct2D (
            td::types::Position a,
            td::types::Position b,
            td::types::Position c )
```

Find the 2D cross product of 2 line segments made by points a, b & c.

Parameters

a	Position of point a
b	Position of point b
c	Position of point c

Returns

2D cross product of line segments ab and bc as given by the formula ab x bc = (b1-a1)∗(c2-b2) - (b2-a2)∗(c1-b1)

8.5.2.3 DotProduct2D()

```
double td::DotProduct2D (
            td::types::Position a,
            td::types::Position b,
            td::types::Position c )
```

Find the 2D dot product of 2 line segments made by points a, b & c.

Parameters

a	Position of point a
b	Position of point b
c	Position of point c

Returns

2D dot product of line segments ab to bc as given by the formula ab.bc = (b1-a1)∗(c1-b1) + (b2-a2)∗(c2-b2)

8.5.2.4 EuclideanDistance()

```
double td::EuclideanDistance (
            td::types::Position p1,
            td::types::Position p2 )
```

Function to calculate the distance between two points 1 and 2.

Parameters

p1	Position of point 1
p1	Position of point 2

Returns

Euclidean distance between point 1 and 2 using distance formula

8.5.2.5 IsCircleCenterInsidePolygon()

```
bool td::IsCircleCenterInsidePolygon (
            td::types::Position p,
            std::vector< std::pair< td::types::Position, td::types::Position > > edges )
```

Function to check if the circle center is inside a polygon or not.

Parameters

p	Position of the center of circle
edges	Vector of all edges that make up the polygon

Returns

True if the circle center is inside the polygon, false otherwise

8.5.2.6 IsCircleCollidingWithCircle()

```
bool td::IsCircleCollidingWithCircle (
            td::types::Position p1,
            float r1,
            td::types::Position p2,
            float r2 )
```

Function to check if a circle is colliding with another circle or not.

Parameters

p1	Position of the center of circle 1
r1	Radius of the circle 1
p2	Position of the center of circle 2
r2	Radius of the circle 2

Returns

True if the two circles are colliding, false otherwise

8.5.2.7 IsCircleCollidingWithPolygon()

```
bool td::IsCircleCollidingWithPolygon (
            td::types::Position p,
            float r,
            std::vector< td::types::Position > polygon_points )
```

Function to check if a circle is colliding with a polygon or not.

Parameters

p	Position of the center of circle
r	Radius of the circle
polygon_points	Vector of all corner points that make up the polygon

Returns

True if the circle is colliding with the polygon, false otherwise

8.5.2.8 IsCircleIntersectingLineSegment()

```
bool td::IsCircleIntersectingLineSegment (
            td::types::Position p,
            float r,
            std::pair< td::types::Position, td::types::Position > line_segment )
```

Check if a given circle intersects a given line segment or not.

Parameters

p	Position of the center of circle
r	Radius of the circle
line_segment	Line segment containing two end points

Returns

True if the circle intersects the line segment, false otherwise

Given a line segment and a circle, there are 3 cases:

1. The circle center is closer to one end point of line segment
2. The circle center is closer to the other end point of line segment
3. The circle center is closer to another point that lies on line segment The cases 1, 2 occur when the dot product of the respective line segment (starting with the respective end point) to the circle center is > 0. Case 3 occurs otherwise. In each case, the closest distance is computed and checked if it is less than or equal to the circle radius.

   DotProduct2D(AB, p) > 0 => p is closest to B

   Check if distance to B <= hitbox of circle(p, r)

   DotProduct2D(BA, p) > 0 => p is closest to A

   Check if distance to A <= hitbox of circle(p, r)

   p is nearest to a point ON segment AB, distance = (AB x Ap)/|AB|

   DotProduct2D(AB, p) > 0 => p is closest to B

   Check if distance to B <= hitbox of circle(p, r)

   DotProduct2D(BA, p) > 0 => p is closest to A

   Check if distance to A <= hitbox of circle(p, r)

   p is nearest to a point ON segment AB, distance = (AB x Ap)/|AB|

8.5.2.9 IsPointBetween()

```
bool td::IsPointBetween (
            td::types::Position a,
            td::types::Position p,
            td::types::Position b )
```

Check if a given point p is between two points a and b.

Parameters

a	Position of point a (end-point of segment ab)
p	Position of the test point p
b	Position of point b (end-point of segment ab)

Returns

True if point p lies on the segment formed by a & b, false otherwise

8.6 collision.hpp

Go to the documentation of this file.

```


#pragma once
#include <cmath>

#include "types.hpp"

namespace td {
constexpr double PI = 3.14;

double EuclideanDistance(td::types::Position p1, td::types::Position p2);

bool IsPointBetween(td::types::Position a, td::types::Position p,
                    td::types::Position b);

double Angle2D(double x1, double y1, double x2, double y2);

double DotProduct2D(td::types::Position a, td::types::Position b,
                    td::types::Position c);

double CrossProduct2D(td::types::Position a, td::types::Position b,
                      td::types::Position c);

bool IsCircleIntersectingLineSegment(
    td::types::Position p, float r,
    std::pair<td::types::Position, td::types::Position> line_segment);

bool IsCircleCenterInsidePolygon(
    td::types::Position p,
    std::vector<std::pair<td::types::Position, td::types::Position» edges);

bool IsCircleCollidingWithCircle(td::types::Position p1, float r1,
                                 td::types::Position p2, float r2);

bool IsCircleCollidingWithPolygon(
    td::types::Position p, float r,
    std::vector<td::types::Position> polygon_points);

}  // namespace td
```

8.7 constants.hpp

```
#pragma once

namespace td {

constexpr unsigned int kCostBasicTower = 100;
constexpr unsigned int kCostBombTower = 200;
constexpr unsigned int kCostSlowingTower = 200;
constexpr unsigned int kCostThornEruptor = 250;
constexpr unsigned int kCostHighDamageTower = 400;
constexpr unsigned int kCostMeltingTower = 150;


}  //namespace td
```

8.8 enemy.hpp

```cpp
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "object.hpp"
#include "projectile.hpp"

namespace td {
//class Map {
//      public:
//      const std::vector<types::Position>& getEnemyPath() const;
//};
class Game;
// {
//      public:
//      const Map* getMap() const;
//};

class Enemy : public Object {
 public:
  Enemy(types::Position position, float hitbox, sf::Texture* texture,
        float health = 100.0f, int move_speed = 1, int bounty = 0,
        bool armored = false, float distance_moved = 0.0f,
        unsigned int slowed_level = 0);

  Enemy(const Enemy& enemy);

  virtual void Update(types::Time dt, const Game& game) {}

  void Update(types::Time dt, const std::vector<types::Position>& path);

  Enemy createBasicCockroach(types::Position startOfTheMap,
                              sf::Texture* texture);

  Enemy createFly(types::Position startOfTheMap, sf::Texture* texture);

  Enemy createBeetle(types::Position startOfTheMap, sf::Texture* texture);

  Enemy createDragonfly(types::Position startOfTheMap,
                                sf::Texture* texture);

  float getHealth() const;

  void setHealth(float health_decrease);

  float getMaxHealth() const;

  int getMoveSpeed() const;

  int getBounty() const;

  bool isArmored() const;

  bool isAtEndOfPath() const;

  void setDistanceMoved(float distance);

  float getDistanceMoved() const;

  void setSlowedLevel(unsigned int level);

  unsigned int getSlowedLevel() const;

  bool TakeDamage(float damage, bool is_armor_piercing);

 protected:
  float health_;
  float max_health_;
  int move_speed_;
  int bounty_;
  bool armored_;
  float distance_moved_;
  unsigned int slowed_level_;
  bool at_end_of_path_;
};
}  // namespace td
```

8.9 game.hpp

```cpp
#pragma once
```

```

#include <list>
#include <map>
#include <nlohmann/json.hpp>
#include <vector>

#include "basic_tower.hpp"
#include "bomb_tower.hpp"
#include "collision.hpp"
#include "enemy.hpp"
#include "high_damage_tower.hpp"
#include "map.hpp"
#include "melting_tower.hpp"
#include "projectile.hpp"
#include "slowing_tower.hpp"
#include "thorn_eruptor.hpp"
#include "tower.hpp"

namespace td {
class Game {
 public:
  Game(Map* map, const std::string& round_file_path,
       const std::map<std::string, sf::Texture*>& textures);

  Game(Map* map, const std::string& round_file_path, int starting_money,
       int starting_lives, const std::map<std::string, sf::Texture*>& textures);

  int getMoney() const;

  int getLives() const;

  void Update();

  std::list<Enemy>& getEnemies();
  const std::list<Enemy>& getEnemies() const;

  std::list<Tower>& getTowers();
  const std::list<Tower>& getTowers() const;

  std::list<Projectile>& getProjectiles();
  const std::list<Projectile>& getProjectiles() const;

  bool getAutoStart() const;

  void setAutoStart(bool auto_start);

  bool SpawnEnemy(const std::string& enemy_identifier,
                  types::Position position);
  bool AddEnemy(const std::string& enemy_identifier, Enemy enemy);

  void AddTower(td::Tower& tower);

  const std::map<const Enemy*, std::vector<const Projectile*»&
  getEnemyCollisions(bool previous_update = false);
  const std::map<const Projectile*, std::vector<const Enemy*»&
  getProjectileCollisions(bool previous_update = false);

  const Map* getMap() const;
  Map* getMap();

  struct Wave {
    std::string enemy_identifier;
    unsigned int spacing = 500;
    unsigned int offset = 0;
    unsigned int count = 1;
    unsigned int enemies_spawned = 0;
    int last_spawn_time = 0;

    Wave(std::string enemy_identifier, unsigned int spacing,
         unsigned int offset, unsigned int count)
        : enemy_identifier(enemy_identifier),
          spacing(spacing),
          offset(offset),
          count(count),
          last_spawn_time(offset-spacing) {}
  };

  void UpgradeTower(Tower* tower);

  void SellTower(Tower* tower);

  std::optional<Tower> StartBuyingTower(std::string name, sf::Texture* tower_texture,
                         sf::Texture* projectile_texture,
                         sf::Texture* extra_texture = nullptr);

  const std::vector<std::vector<Wave»& getRounds();

  void LoadRounds(const std::string& file_path);

  bool CheckTowerPlacementCollision(const Tower& tower);

  void StartRound(size_t round_index);

  bool IsRoundInProgress();

  size_t getCurrentRoundIndex();

  size_t getMaxRoundIndex();

  void Unpause();

 private:
  void LoadEnemies(const std::map<std::string, sf::Texture*>& textures);

  unsigned int money_;
  int lives_;
  std::list<Enemy> enemies_;
  std::list<Tower> towers_;
  std::list<Projectile> projectiles_;
  std::map<const Enemy*, std::vector<const Projectile*» enemy_collisions_;
  std::map<const Enemy*, std::vector<const Projectile*»
      previous_enemy_collisions_;
  std::map<const Projectile*, std::vector<const Enemy*» projectile_collisions_;
  std::map<const Projectile*, std::vector<const Enemy*»
      previous_projectile_collisions_;
  std::map<std::string, Enemy> enemy_table_;
  std::vector<std::vector<Wave» rounds_;
  Map* map_;
  sf::Clock update_clock_;
  unsigned int round_time_;
  size_t current_round_index_;
  bool round_in_progress_;
  bool auto_start_;
};
}  // namespace td
```

8.10 high_damage_tower.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"

namespace td {

class High_damage_tower : public Tower {
 public:
  High_damage_tower(types::Position position, float rotation_angle = 0.0f,
                    sf::Texture* texture = nullptr,
                    sf::Texture* texture_projectile = nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  bool Shoot(std::list<Projectile>&, std::list<Enemy>& enemies);
};
}  // namespace td
```

8.11 map.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <nlohmann/json.hpp>
#include <vector>

#include "types.hpp"

namespace td {
class Map {
 public:
  Map(const std::string& background_image_path,
      std::vector<td::types::Position> enemy_path,
      std::vector<td::types::BlockedRegion> blocked_regions);

  const std::string& getBackgroundImagePath();

  std::vector<td::types::Position>& getEnemyPath();
  const std::vector<td::types::Position>& getEnemyPath() const;

  std::vector<td::types::BlockedRegion>& getBlockedRegions();
  const std::vector<td::types::BlockedRegion>& getBlockedRegions() const;

  td::types::Position GetStartingPosition();

  static Map* LoadFromFile(const std::string& file_name);

 private:
  const std::string& background_image_path_;
  std::vector<td::types::Position> enemy_path_;
  std::vector<td::types::BlockedRegion> blocked_regions_;
};
}  // namespace td
```

8.12 melting_tower.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"

namespace td {
class Melting_tower : public Tower {
 public:
  Melting_tower(sf::Vector2<float> position, float rotation_angle = 0.0f, sf::Texture* texture =
    nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  bool Shoot(std::list<Projectile>&, std::list<Enemy>& enemies);
};
}  // namespace td
```

8.13 object.hpp

```
#pragma once

#include "types.hpp"

namespace td {
class Game;
class Object {
 public:
  Object(types::Position position = types::Position(0.0f, 0.0f),
         float hitboxRadius = 0.0f, sf::Texture* texture = nullptr,
         float rotation_angle = 0.0f);

  Object(const Object& obj);

  virtual types::Position getPosition() const;

  virtual float getHitboxRadius() const;

  virtual const sf::Texture* getTexture() const;

  virtual sf::Texture* getTexture();

  virtual void setPosition(types::Position position);

  virtual void setRotation(float angle);

  virtual float getRotation() const;

  void Delete();

  bool IsDeleted();

 protected:
  types::Position position_;
  float
      hitboxRadius_;
  sf::Texture* texture_;
  float rotation_angle_;

 private:
  bool preserve_;
};
}  // namespace td
```

8.14 projectile.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "enemy.hpp"
#include "object.hpp"

namespace td {
class Game;
class Enemy;
class Projectile : public Object {
 public:
  Projectile(types::Position position, float hitbox, sf::Texture* texture,
             float rotation_angle, float damage, bool is_armor_piercing,
             unsigned int piercing, float speed, float lifetime);

  virtual void Update(types::Time dt, std::list<Enemy>& enemies,
                      std::list<Projectile>& projectiles);

  void Update(types::Time dt);

  float getDamage() const;

  bool isArmorPiercing() const;

  void setPiercingLeft(unsigned int count);

  float getSpeed() const;

  float getLifetimeLeft() const;

  unsigned int getPiercingLeft() const;

 protected:
  float damage_;
  bool is_armor_piercing_;
  unsigned int piercing_left_;
  float speed_;
  float lifetime_left_;
};
}  // namespace td
```

8.15 slowing_tower.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"

namespace td {
class Slowing_tower : public Tower {
 public:
  Slowing_tower(sf::Vector2<float> position, float rotation_angle = 0.0f, sf::Texture* texture =
    nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  bool Shoot(std::list<Projectile>&, std::list<Enemy>& enemies);
};
}  // namespace td
```

8.16 splitworm.hpp

```
#pragma once

#include <SFML/Graphics.hpp>

#include "enemy.hpp"

namespace td {
class Splitworm : public Enemy {
 public:
  Splitworm(types::Position startOfTheMap, float hitbox, sf::Texture* texture,
            float health = 150, int move_speed = 20, float bounty = 0.0f,
            bool armored = true, float distance_moved = 0.0f);


  std::vector<Enemy> doUponDeath(sf::Texture* texture);

};
}  // namespace td
```

8.17 thorn_eruptor.hpp

```
#pragma once

#include <SFML/Graphics.hpp>
#include <list>

#include "tower.hpp"
#include "collision.hpp"


namespace td {
class ThornEruptor : public Tower {
 public:
  ThornEruptor(types::Position position, float rotation_angle = 0.0f, sf::Texture* texture = nullptr,
    sf::Texture* texture_projectile = nullptr);

  void Upgrade();

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  bool Shoot(std::list<Projectile>& projectiles, std::list<Enemy>& enemies);
};
}  // namespace td
```

8.18 tower.hpp

```
#pragma once

#include <optional>

#include "enemy.hpp"
#include "object.hpp"
#include "projectile.hpp"
#include "types.hpp"
#include "collision.hpp"

namespace td {
class Game;
class Tower : public Object {
 public:
  Tower(types::Position position, float hitbox, sf::Texture* texture,
        sf::Texture* texture_projectile, float rotation_angle = 0.0f,
        unsigned int attack_speed = 1U, float range = 1.0f, unsigned int cost = 0, unsigned int
    upgrade_cost = 0,
        unsigned int level = 1, types::Targeting targetTo = types::kFirst);

  void Update(types::Time dt, const td::Game&);

  void Update(types::Time dt, std::list<Enemy>& enemies, std::list<Projectile>& projectiles);

  explicit Tower(types::Position position, float rotation_angle,
                 unsigned int attack_speed);

  unsigned int getAttackSpeed() const;

  float getRange() const;

unsigned int getLevel() const;

unsigned int getCost() const;

unsigned int getMoneySpent() const;

void setMoneySpent(unsigned int value);

unsigned int getUpgradeCost() const;

const std::string& getName() const;

types::Targeting getTargeting() const;

void setTargeting(types::Targeting targeting);

virtual bool Shoot(std::list<Projectile>& projectiles,
                                     std::list<Enemy>& enemies);

virtual void Upgrade() { level_++; };

virtual std::optional<Enemy*> GetTarget(
        std::list<Enemy>& enemies);

 types::Position GetProjectStartPos();

protected:
 std::string name_;
 unsigned int attack_speed_;
 float range_;
 unsigned int level_;
 types::Targeting targeting_;
 sf::Texture* texture_projectile_;
 unsigned int cost_;
 unsigned int upgrade_cost_;
 unsigned int money_spent_on_tower_;
 types::Time time_since_last_shoot_;
};
}  // namespace td
```

8.19 types.hpp

```
#pragma once

#include <SFML/Graphics.hpp>

namespace td::types {
using Position =
    sf::Vector2f;
using Time = sf::Time;
enum Targeting { kFirst, kLast, kClose, kStrong, kArea };

enum AppState { kMainMenu, kOptions, kMapSelect, kGame, kPause, kUpgrade };
using BlockedRegion =
    sf::ConvexShape;
}  // namespace td::types
```

Index