Orcs n Towers

Generated by Doxygen 1.9.1

Chapter 1

Orcs n Towers

1.1 Overview

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

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

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

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

1.2 Instructions

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

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

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

1.3 How to compile the program

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

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

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

1.4 Testing

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

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

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

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

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

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

1.5 Work log

1.5.0.0.1 Division of work / main responisbilities:

Pavel Filippov:

- Tower class and it's derived classes (bullet-, bomb-, missile-, poision-, and freezing tower)
- Game class

Otto Litkey:

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

Ellen Molin:

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

Leo Saied-Ahmad:

- Enemy class

Tuan Vu:

- Path

1.5.0.0.2 Weekly breakdown

Week 1 Pavel Filippov:

- Initialised implementation of base tower class.

Otto Litkey:

- Initialised implementation of class(es) resposible for graphics.

Ellen Molin:

- Initialised implementation of Player and base Projectile classes.

Leo Saied-Ahmad:

- Intialised implementation of base Enemy class.

Tuan Vu:

- Initialised implementation of Map class.

Week 2 Pavel Filippov:

- Continued implementation of base tower class.

Otto Litkey:

- Initialised game class and ResourceContainer template class.

Ellen Molin:

- Continued implementation of Player and base Projectile classes.

Leo Saied-Ahmad:

- Continued implementation of base Enemy class.

Tuan Vu:

- Continued implementation of Map class.

Week 3 Pavel Filippov:

- Created update function in game class, as well as for towers.

Otto Litkey:

- Tested rendering, beginnings of dragging and dropping functionality for creating towers.

Ellen Molin:

- Improved projectile class and added functionality.

Leo Saied-Ahmad:

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

Tuan Vu:

- Finished implementing loading map from file, worked on drawing and being able to sell towers

Week 4 Pavel Filippov:

- Created derived classes bulletTower and bombTower from tower. Moved tower update logic to it's own function

Otto Litkey:

- Continued testing rendering, finished drag and drop functionality for creating towers, and added way to pause game.

Ellen Molin:

- Created derived classes bullet and bomb from projectile. Improved projectile methods to better work with derived classes.

Leo Saied-Ahmad:

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

Tuan Vu:

- Continued working on map class to create a path.

Week 5 Pavel Filippov:

- Implemented freezing effect tower, poision effect tower and missile tower.

Otto Litkey:

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

Ellen Molin:

- Implemented a levelManager that handles creating and managing levels. Created a missile projectile that follows enemy.

Leo Saied-Ahmad:

- Implemented slowing effect on enemies and refiened enemy movement. Worked on graphics: show players state, end screen when player loses.

Tuan Vu:

- Implemented functionality so that towers can't be built on the path

Chapter 2

Source content

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

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Class Documentation

5.1 BombProjectile Class Reference

a projectile that causes damage to multiple enemies

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

Inheritance diagram for BombProjectile:

classBombProjectile-eps-converted-to.pdf

Public Member Functions

- BombProjectile (sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range)
- bool hasHitEnemy (std::shared_ptr< Enemy > &enemy) override
- void update (Game &game) override
- Textures::ProjectileID textureType () override

Private Attributes

- int **blastRange_**

5.1.1 Detailed Description

a projectile that causes damage to multiple enemies

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BombProjectile()

```
BombProjectile::BombProjectile (
             sf::Vector2f shootDirection,
             sf::Vector2f position,
             int damage,
             float range )  [inline]
```

Parameters

*blast←↩ Range_*	the blast radius of the bomb

5.1.3 Member Function Documentation

5.1.3.1 hasHitEnemy()

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

Calculates the distance between the bomb and an enemy. If the enemy is within the blast range, cause damage to it because it has been hit.

Returns

true if bomb has hit an enemy.

Parameters

enemy	is a reference to an Enemy object

Implements Projectile.

5.1.3.2 textureType()

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

Returns

the texture ID of the type this derived class uses

Implements Projectile.

5.1.3.3 update()

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

If the bomb has reached it's maximum distance, it goes through all the enemies in the game to see if it hits any, and once done with that, is destroyed. If the bomb hasn't yet reached it's maximum distance, it is moved.

Parameters

game	is a reference to the running game instance

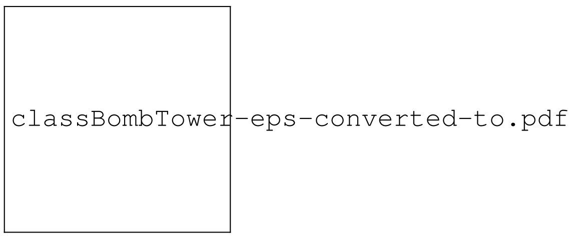
Implements Projectile.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bombProjectile.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bombProjectile.cpp

5.2 BombTower Class Reference

Inheritance diagram for BombTower:



Public Member Functions

- **BombTower** (sf::Vector2f)
- void update (std::list< std::shared_ptr< Enemy >> &enemies, sf::Time time) override

    update() method is virtual as some types of towers use base update()
- BombProjectile ∗ shoot () override

    shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

5.2.1 Member Function Documentation

5.2.1.1 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile*

Implements Tower.

5.2.1.2 update()

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

update() method is virtual as some types of towers use base update()

Main tower logic.

Parameters

enemies	is the list of Enemy shared pointers needed for setting lockedEnemy_
time	is delta time from Game object needed for updating

See also

fireTimer_

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

See also

lockedEnemy_ member stays nullptr. Otherwise, lockedEnemy_ is set to the pointer to the fastest, alive enemy within tower's range.

Parameters

enemies	is passed from calling

See also

Game::update method

Parameters

time	is passed from calling

See also

Game::update method and is used to update

fireTimer_

Reimplemented from Tower.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bombTower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bombTower.cpp

5.3 BulletProjectile Class Reference

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

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

Inheritance diagram for BulletProjectile:

```
classBulletProjectile-eps-converted-to.pdf
```

Public Member Functions

- **BulletProjectile** (sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range)
- bool hasHitEnemy (std::shared_ptr< Enemy > &enemy) override

  Checks if the bullet has hit an enemy. If the bullet's and enemy's sprites intersect, there has been a hit and the bullet causes damage to the enemy.
- void update (Game &game) override
- Textures::ProjectileID textureType () override
- float rotationAngle () const

  Calculates the rotation angle of the bullet based on its shooting direction !!! what is it used for.

5.3.1 Detailed Description

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

5.3.2 Member Function Documentation

5.3.2.1 hasHitEnemy()

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

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

Returns

true if bullet has hit an enemy.

Parameters

enemy	is a reference to an Enemy object

Implements Projectile.

5.3.2.2 textureType()

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

Returns

the texture ID of the type this derived class uses.

Implements Projectile.

5.3.2.3 update()

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

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

Parameters

game	is a reference to the running game instance

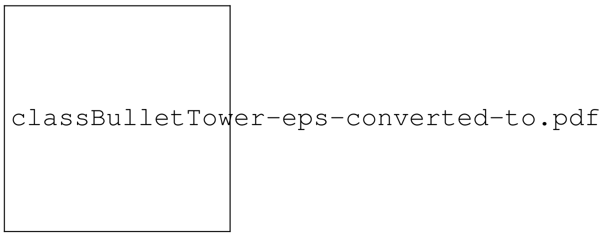
Implements Projectile.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bulletProjectile.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bulletProjectile.cpp

5.4 BulletTower Class Reference

Inheritance diagram for BulletTower:



Public Member Functions

- BulletTower (sf::Vector2f)

  Constructs a new Bullet Tower object.
- BulletProjectile ∗ shoot () override

  shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

5.4.1 Constructor & Destructor Documentation

5.4.1.1 BulletTower()

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

Constructs a new Bullet Tower object.

Parameters

position	is mouse position passed by the caller.

5.4.2 Member Function Documentation

5.4.2.1 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile∗

Implements Tower.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bulletTower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/bulletTower.cpp

5.5 Button Class Reference

Inheritance diagram for Button:

Public Member Functions

- **Button** (Actions action, sf::Texture &texture, sf::Vector2f position, std::string text, sf::Font &font)
- bool **isClicked** (sf::Vector2f mousePos) const
- Actions **getAction** () const
- sf::Text **getLabel** () const

Private Attributes

- Actions **action_**
- sf::Text **label_**

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/button.hpp

5.6 Enemy Class Reference

Inheritance diagram for Enemy:

classEnemy-eps-converted-to.pdf

Public Member Functions

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

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

  //damages the enemy, takes in a damage value as a parameter, if the damage is higher than the health the enemy is a automatically killed
- void kill ()

  kills the enemy, sets dead variable to true
- void applyPoison (int stacksOfPoison, int damagePerStack)

  applies poison status effect to enemies
- void applySlowed (sf::Time duration, float slowCoefficient)

  applies slowed status effect to enemies
- void **slowedDamage** ()
- void setVelocity ()

  sets the enemy velocity based on where the current waypoint is
- bool isWaypointPassed (sf::Vector2f movement)

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

  finds a newwaypoint for the enemy, this function goes through the waypoints qeue and sets the current waypoint as the next waypoint in the qeue if waypoints are empty it means the enemy has reached the castle and the enemy is set to state dead
- std::queue< sf::Vector2f > getWaypoints ()
- void **moveEnemy** (sf::Vector2f movement)
- int getMoney () const
- void updateHealthText (const sf::Font &font)

  updates the health text above enemies with the enemies current health
- const sf::Text & getHealthText () const

Private Attributes

- int **hp_**
- int **initialHp_**
- bool **dead_** = false
- float **speed_**
- float **actualSpeed_**
- float **effectiveSpeed_**
- sf::Text **healthText_**
- EnemyType **type_**
- int **poison_** = 0
- sf::Time **slowed_** = sf::Time::Zero
- int **money_**
- sf::Vector2f **velocity_**
- std::queue< sf::Vector2f > **waypoints_**
- sf::Vector2f **currentWaypoint_**
- int **direction_**
- int **poisonDamage** = 0
- sf::Time **poisonTimer_**
- float **slowCoefficient_** = 0.f

5.6.1 Constructor & Destructor Documentation

5.6.1.1 Enemy()

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

Initialises an enemy

Parameters

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

5.6.2 Member Function Documentation

5.6.2.1 dead()

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

Returns

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

5.6.2.2 getCenter()

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

Returns

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

5.6.2.3 getHealthText()

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

Returns

returns the healthText

5.6.2.4 getLocation()

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

Returns

returns the enemies location as a sf::Vector2f

5.6.2.5 getMoney()

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

Returns

returns the amount of money this enemy provides when killed

5.6.2.6 getWaypoints()

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

Returns

returns waypoints

5.6.2.7 hp()

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

Returns

returns enemy hp

5.6.2.8 initialHp()

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

Returns

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

5.6.2.9 isWaypointPassed()

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

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

Returns

returns a bool

5.6.2.10 poisonStatus()

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

Returns

returns the duration of poison status effect

5.6.2.11 slowedStatus()

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

Returns

returns the duration of slowed status effect

5.6.2.12 speed()

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

Returns

returns enemies speed

5.6.2.13 type()

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

Returns

returns enemy type

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/enemy.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/enemy.cpp

5.7 Explosion Class Reference

Small class for drawing bomb explosions.

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

Inheritance diagram for Explosion:

classExplosion-eps-converted-to.pdf

Public Member Functions

- Explosion (int blastRange, sf::Vector2f pos)

  Constructs an explosion.
- void update (sf::Time inputtime)

  Updates the explosion.
- bool isDone ()

  Return done_ which tells if the explosion is done.

Private Attributes

- sf::Time **time_**
- int **blastRange_**
- bool **done_**

5.7.1 Detailed Description

Small class for drawing bomb explosions.

See also

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Explosion()

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

Constructs an explosion.

Parameters

blastRange	Stores the bomb's blast range
pos	The bomb's position

5.7.3 Member Function Documentation

5.7.3.1 update()

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

Updates the explosion.

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

Parameters

inputtime	Time between frames from Game::getTime()

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/explosion.hpp

5.8 FreezingTower Class Reference

Inheritance diagram for FreezingTower:

classFreezingTower-eps-converted-to.pdf

Public Member Functions

- **FreezingTower** (sf::Vector2f)
- void update (std::list< std::shared_ptr< Enemy >> &enemies, sf::Time time) override

  update() method is virtual as some types of towers use base update()
- Projectile $*$ shoot () override

  shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).
- void upgradeTower () override

  upgradeTower() method is virtual as upgrade logic is same for all types of towers except Freezing Tower

Private Attributes

- std::list< std::shared_ptr< Enemy > > **lockedEnemies_**
- float **slowCoefficient_** = 0.1

5.8.1 Member Function Documentation

5.8.1.1 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile*

Implements Tower.

5.8.1.2 update()

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

update() method is virtual as some types of towers use base update()

Main tower logic.

Parameters

enemies	is the list of Enemy shared pointers needed for setting lockedEnemy_
time	is delta time from Game object needed for updating

See also

fireTimer_

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

See also

lockedEnemy_ member stays nullptr. Otherwise, lockedEnemy_ is set to the pointer to the fastest, alive enemy within tower's range.

Parameters

enemies	is passed from calling

See also

Game::update method

Parameters

time	is passed from calling

See also

Game::update method and is used to update

fireTimer_

Reimplemented from Tower.

5.8.1.3 upgradeTower()

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

upgradeTower() method is virtual as upgrade logic is same for all types of towers except Freezing Tower

Upgrade the tower to the next level.

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

Note

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

Reimplemented from Tower.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/freezingTower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/freezingTower.cpp

5.9 Game Class Reference

This class runs the game logic.

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

Public Member Functions

- void run ()

  this function is called from the main function to run the game.
- path & getPath ()

  Returns the path, which enemies follow.

Public Attributes

- Map **map**

Private Member Functions

- void processEvents ()

  processes user input
- void **update** ()
- void **render** ()
- void **loadTextures** ()
- void **createPath** ()
- void **checkTowers** ()
- void **testEnemy** ()
- void **testEnemySplit** (sf::Vector2f position, std::queue< sf::Vector2f > waypoints)
- void **updateMenus** ()
- sf::Time **getTime** () const

Private Attributes

- sf::Clock **clock_**
- sf::Time **time_**
- sf::RenderWindow **window_**
- std::list< Tower * > **towers_**
- std::list< std::shared_ptr< Enemy > > **enemies_**
- std::list< Projectile * > **projectiles_**
- std::list< Explosion * > **explosions_**
- path **path_**
- std::list< Button > **buttons_**
- bool **dragged_**
- bool **paused_**
- bool **isGameOver_** =false
- bool **isGameFinished_** = false
- sf::Font **font_**
- sf::Text **gameOverText**
- sf::Text **gameFinishedText**
- sf::Sprite **castle_sprite_**

- std::unique_ptr< Menu > **shop_**
- std::unique_ptr< Menu > **alternativeMenu_**
- Tower ∗ **activeTower_**
- bool **menuInactive** = false
- ResourceContainer< Textures::TowerID, sf::Texture > **tower_textures_**
- ResourceContainer< Textures::EnemyID, sf::Texture > **enemy_textures_**
- ResourceContainer< Textures::ProjectileID, sf::Texture > **projectile_textures_**
- ResourceContainer< Textures::Various, sf::Texture > **various_textures_**
- Player **player_**
- LevelManager **levelManager_**

Friends

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

5.9.1 Detailed Description

This class runs the game logic.

5.9.2 Member Function Documentation

5.9.2.1 getPath()

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

Returns the path, which enemies follow.

Returns

path& the path

5.9.2.2 processEvents()

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

processes user input

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

5.9.2.3 run()

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

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

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

See also

processEvents()
update()
render()

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/game.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/game.cpp

5.10 LevelManager Class Reference

Handles the creation and managing of levels.

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

Public Types

- using variantData = std::variant< int, float, std::vector< int > >

  To allow the map holding level information to use different types.

Public Member Functions

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

Private Member Functions

- void readLevels ()
- void initiateEnemies ()

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

Private Attributes

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

  Container to hold all the levels One entry in the outer container (vector) is one level, meaning index 0 is level one. The inner map holds all information regarding the specific level.
  Keys:
  .
- int **currLevel_**
- const std::string & **src_**
- bool **readingSuccess_**
- int **levelTotal_**
- float **waitTime_**
- path & **path_**
- Game & **game_**
- Player & **player_**

5.10.1 Detailed Description

Handles the creation and managing of levels.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 LevelManager()

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

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

Parameters

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

See also

readLevels()

5.10.3 Member Function Documentation

5.10.3.1 getCurrentLevel()

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

Returns

the current level

5.10.3.2 getLevelTotal()

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

Returns

the total number of levels definend

5.10.3.3 readingSuccessfull()

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

returns status flag for reading level info from file.

Returns

True if reading was successfull, false if not

5.10.3.4 readLevels()

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

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

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

5.10.3.5 update()

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

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

See also

initiateEnemies()

5.10.4 Member Data Documentation

5.10.4.1 levelSpecs_

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

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

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

  See also

  Enemy class' type enum
  variantData The container that stores all level information

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/levelManager.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/levelManager.cpp

5.11 Map Class Reference

Inheritance diagram for Map:

Public Member Functions

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

Public Attributes

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

Private Member Functions

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

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/map.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/map.cpp

5.12 Menu Class Reference

Class for storing a collection of buttons, a menu.

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

Public Member Functions

- void draw (sf::RenderWindow &window)

  Draws all the objects in the menu.
- void checkButtons (Game ∗game)

  Checks if a button in the menu has been pressed.
- void createMenu (MenuType menu, Game ∗game)

  Creates the buttons and texts of a menu.
- void update (Player &player)

  Updates the status of the menu.
- void drag (Game ∗game)

  Implements drag&drop placing of towers.
- void drawRange (Game ∗game)

  Draws active tower range.

Private Member Functions

- void newTower (Tower ∗tower, Game ∗game)

  Adds a new tower to the game, called in checkButtons.
- bool canBePlaced (Game ∗game)

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

Private Attributes

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

5.12.1 Detailed Description

Class for storing a collection of buttons, a menu.

5.12.2 Member Function Documentation

5.12.2.1 canBePlaced()

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

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

Parameters

game	Pointer to the game object

Returns

true, if the tower can be placed

5.12.2.2 checkButtons()

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

Checks if a button in the menu has been pressed.

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

Parameters

game	Pointer to the game object

5.12.2.3 createMenu()

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

Creates the buttons and texts of a menu.

Parameters

menu	Enumerator which tells the type of menu being created
game	Poiner to the game object

5.12.2.4 drag()

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

Implements drag&drop placing of towers.

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

Parameters

game	pointer to the game object

See also

canBePlaced()

5.12.2.5 draw()

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

Draws all the objects in the menu.

Parameters

window	window onto which the objects get drawn

5.12.2.6 drawRange()

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

Draws active tower range.

Parameters

game	pointer to the game object

5.12.2.7 newTower()

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

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

Parameters

tower	Pointer to new tower being built
game	Pointer to game

5.12.2.8 update()

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

Updates the status of the menu.

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

Parameters

player	Reference to the player object

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/menu.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/menu.cpp

5.13 MissileProjectile Class Reference

A projectile that targets (follows) a specific enemy.

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

Inheritance diagram for MissileProjectile:

classMissileProjectile-eps-converted-to.pdf

Public Member Functions

- MissileProjectile (sf::Vector2f position, int damage, std::shared_ptr< Enemy > targetEnemy)
- bool hasHitEnemy (std::shared_ptr< Enemy > &enemy) override
- void update (Game &game) override
- Textures::ProjectileID textureType () override

Private Attributes

- std::shared_ptr< Enemy > **targetEnemy_**

5.13.1 Detailed Description

A projectile that targets (follows) a specific enemy.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 MissileProjectile()

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

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

Parameters

targetEnemy	is the enemy that the missile is targeting (following).

5.13.3 Member Function Documentation

5.13.3.1 hasHitEnemy()

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

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

Returns

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

Parameters

enemy	is a reference to an Enemy object, the missiles target.

Implements Projectile.

5.13.3.2 textureType()

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

Returns

the texture ID of the type this derived class uses.

Implements Projectile.

5.13.3.3 update()

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

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

Parameters

game	is a reference to the running game instance.

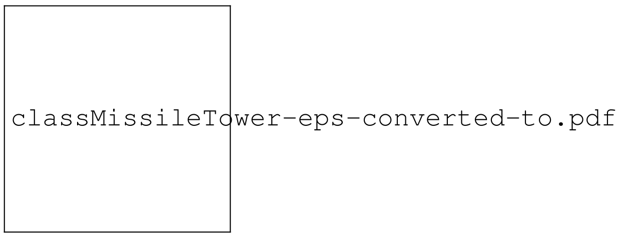
Implements Projectile.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/missileProjectile.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/missileProjectile.cpp

5.14 MissileTower Class Reference

Inheritance diagram for MissileTower:



Public Member Functions

- **MissileTower** (sf::Vector2f)
- MissileProjectile ∗ shoot () override

    shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

5.14.1 Member Function Documentation

5.14.1.1 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile∗

Implements Tower.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/missileTower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/missileTower.cpp

5.15 path Class Reference

Public Member Functions

- path (const std::string &src)
- void readPath ()
- bool readingSuccessfull ()
- void **addWaypoint** (const sf::Vector2f &point)
- std::queue< sf::Vector2f > **getWaypoints** () const
- void **makeUnBuildablePath** ()

Public Attributes

- std::queue< sf::Vector2f > **waypoints_**
- std::vector< sf::Vector2f > **wayPoints**
- std::vector< sf::FloatRect > **unBuildable**
- std::vector< std::vector< sf::Vector2f > > **paths_**

Static Public Attributes

- static const float **width** = 60.f

Private Attributes

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

Friends

- class **enemy**

5.15.1 Constructor & Destructor Documentation

5.15.1.1 path()

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

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

Parameters

src	is the source of the path information file to be read

See also

readPath()

5.15.2 Member Function Documentation

5.15.2.1 readingSuccessfull()

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

Returns

5.15.2.2 readPath()

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

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

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/path.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/path.cpp

5.16 Player Class Reference

Inheritance diagram for Player:

classPlayer-eps-converted-to.pdf

Public Member Functions

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

Private Attributes

- int **hp_**
- int **wallet_**
- int **level_**

5.16.1 Constructor & Destructor Documentation

5.16.1.1 Player()

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

Initialises a player with default values.

Parameters

*hp_*	is the health points of the player
wallet↩ _	is how much money the player has
level↩ _	is the level of the player

5.16.2 Member Function Documentation

5.16.2.1 addMoney()

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

adds money to the players wallet

Parameters

amount	is how much money is to be added

5.16.2.2 getHP()

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

Returns

how many health points the player has

5.16.2.3 getLevel()

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

Returns

the current level of the player

5.16.2.4 getWallet()

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

Returns

how much money the player has

5.16.2.5 removeHP()

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

removes health points from the player

Parameters

amount	is how much hp is to be removed

5.16.2.6 removeMoney()

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

removes money from the players wallet

Parameters

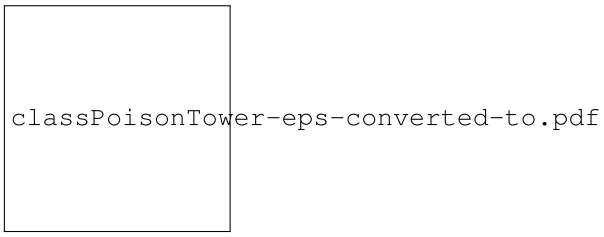
cost	is how much money is to be removed

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/player.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/player.cpp

5.17 PoisonTower Class Reference

Inheritance diagram for PoisonTower:



Public Member Functions

- **PoisonTower** (sf::Vector2f)
- void update (std::list< std::shared_ptr< Enemy >> &enemies, sf::Time time) override

    update() method is virtual as some types of towers use base update()
- Projectile ∗ shoot () override

    shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Private Attributes

- std::list< std::shared_ptr< Enemy > > **lockedEnemies_**

5.17.1 Member Function Documentation

5.17.1.1 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile∗

Implements Tower.

5.17.1.2 update()

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

update() method is virtual as some types of towers use base update()

Main tower logic.

Parameters

enemies	is the list of Enemy shared pointers needed for setting lockedEnemy_
time	is delta time from Game object needed for updating

See also

fireTimer_

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

See also

lockedEnemy_ member stays nullptr. Otherwise, lockedEnemy_ is set to the pointer to the fastest, alive enemy within tower's range.

Parameters

enemies	is passed from calling

See also

Game::update method

Parameters

time	is passed from calling

See also

Game::update method and is used to update

fireTimer_

Reimplemented from Tower.

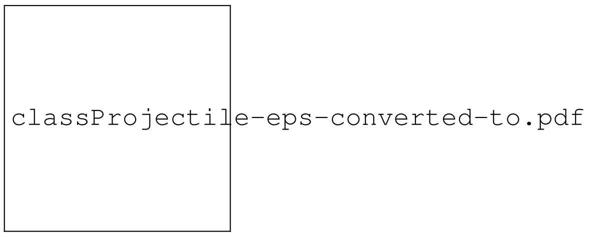
The documentation for this class was generated from the following files:

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/poisonTower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/poisonTower.cpp

5.18 Projectile Class Reference

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

Inheritance diagram for Projectile:



Public Member Functions

- Projectile (sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed, std::string type, float maxDistance)

  Construcs a projectile and sets it's initial position.
- virtual ∼Projectile ()

  Destroy the Projectile object.
- float getSpeed () const
- const std::string & getType () const
- int getDamage () const
- sf::Vector2f getShootDir () const
- void destroy ()
- bool isDestroyed ()

  Returns wheter the projectile is destroyed, and needs to be deleted, or not.
- bool distToTower ()

  Calculates the projectiles distance from the tower that created it.
- virtual bool hasHitEnemy (std::shared_ptr< Enemy > &)=0

  checks if the projectile has hit an enemy. Overridden in each derived class.
- virtual void update (Game &)=0

  updates the projectiles state as is defiened in each derived class
- virtual Textures::ProjectileID textureType ()=0

Private Attributes

- float **speed_**
- std::string **type_**
- int **damage_**
- sf::Vector2f **position_**
- float **maxDistance_**
- sf::Vector2f **shootDirection_**
- bool **isDestroyed_**

5.18.1 Detailed Description

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

5.18.2 Constructor & Destructor Documentation

5.18.2.1 Projectile()

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

Construcs a projectile and sets it's initial position.

Parameters

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

5.18.3 Member Function Documentation

5.18.3.1 destroy()

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

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

5.18.3.2 distToTower()

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

Calculates the projectiles distance from the tower that created it.

Returns

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

5.18.3.3 getDamage()

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

Returns

the damage of the projectile

5.18.3.4 getShootDir()

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

Returns

the directional vector of the projectile

5.18.3.5 getSpeed()

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

Returns

the speed of the projectile

5.18.3.6 getType()

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

Returns

the type of the projectile

5.18.3.7 textureType()

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

Returns

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

Implemented in MissileProjectile, BulletProjectile, and BombProjectile.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/projectile.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/projectile.cpp

5.19 ResourceContainer< T_enum, T_resource > Class Template Reference

Template container for textures etc resources.

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

Public Member Functions

- void load (T_enum type, std::string filename)
  Loads and stores a resource.
- T_resource & get (T_enum type) const
  Find and return requested resource.

Private Attributes

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

5.19.1 Detailed Description

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

Template container for textures etc resources.

5.19.2 Member Function Documentation

5.19.2.1 get()

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

Find and return requested resource.

Parameters

type	Enumerator defining which texture is wanted

Returns

Returns reference to recource if found

5.19.2.2 load()

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

Loads and stores a resource.

Parameters

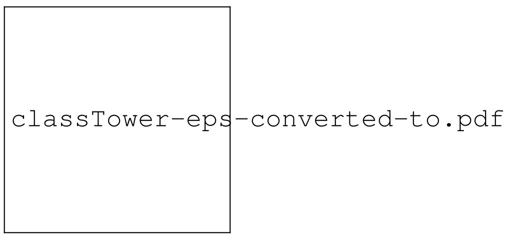
type	Enumerator which defines the type of this resource.
filename	path to file containing the resource.

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/resource_container.hpp

5.20 Tower Class Reference

Inheritance diagram for Tower:



Public Member Functions

- Tower (sf::Vector2f position, const std::string &type, int baseCost, float range, sf::Time fireRate, int damage, int currentLvl, int upgradeCost, std::shared_ptr< Enemy > lockedEnemy, bool maxLevelReached)

  Constructor for abstract tower used in constructor for derived tower classes.
- const std::string & **getType** () const
- const int **getBaseCost** () const
- sf::Time **getFireRate** () const
- const float **getRange** () const
- int **getDamage** () const
- std::shared_ptr< Enemy > **getLockedEnemy** () const
- bool **isMaxLevelReached** () const
- int **getCurrentLvl** () const
- const int **getUpgradeCost** () const
- sf::Time **getFireTimer** ()
- bool enemyWithinRange (std::shared_ptr< Enemy > enemy)

  Check if the enemy is within the range of the tower.
- virtual Projectile ∗ shoot ()=0

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

- virtual void upgradeTower ()

    upgradeTower() method is virtual as upgrade logic is same for all types of towers except Freezing Tower
- virtual void update (std::list< std::shared_ptr< Enemy >> &enemies, sf::Time time)

    update() method is virtual as some types of towers use base update()
- void updateFireTimer (sf::Time &dt)

    Increments fireTimer_ by dt.
- void **setLevel** (int level)
- void **setMaxLevelFlag** ()
- void **setLockedEnemy** (std::shared_ptr< Enemy > enemy)
- void **resetFireTimer** ()

Private Attributes

- const std::string type_

    Private members of abstract tower class.
- const int **baseCost_**
- const float **range_**
- int **damage_**
- int **currentLvl_**
- const int **upgradeCost_**
- std::shared_ptr< Enemy > **lockedEnemy_**
- sf::Time **fireTimer_**
- sf::Time **fireRate_**
- bool **maxLevelReached_**

5.20.1 Constructor & Destructor Documentation

5.20.1.1 Tower()

```
Tower::Tower (
              sf::Vector2f position,
              const std::string & type,
              int baseCost,
              float range,
              sf::Time fireRate,
              int damage,
              int currentLvl,
              int upgradeCost,
              std::shared_ptr< Enemy > lockedEnemy,
              bool maxLevelReached )
```

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

Parameters

position	is determined by constructor of derived tower class
type	is determined by constructor of derived tower class

Parameters

baseCost	is determined by constructor of derived tower class
range	is determined by constructor of derived tower class
fireRate	is determined by constructor of derived tower class
damage	is determined by constructor of derived tower class
currentLvl	is determined by constructor of derived tower class
upgradeCost	is determined by constructor of derived tower class
lockedEnemy	is determined by constructor of derived tower class
maxLevelReached	is determined by constructor of derived tower class

5.20.2 Member Function Documentation

5.20.2.1 enemyWithinRange()

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

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

Parameters

enemy	is passed from calling

See also

Tower::update method

Returns

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

false otherwise

5.20.2.2 shoot()

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

shoot() method is pure virtual as different types of towers produce different types of projectiles (or no projectiles at all as is the case with Poison and Freezing Towers).

Returns

Projectile∗

Implemented in PoisonTower, MissileTower, FreezingTower, BulletTower, and BombTower.

5.20.2.3 update()

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

update() method is virtual as some types of towers use base update()

Main tower logic.

Parameters

enemies	is the list of Enemy shared pointers needed for setting lockedEnemy_
time	is delta time from Game object needed for updating

See also

fireTimer_

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

See also

lockedEnemy_ member stays nullptr. Otherwise, lockedEnemy_ is set to the pointer to the fastest, alive enemy within tower's range.

Parameters

enemies	is passed from calling

See also

Game::update method

Parameters

time	is passed from calling

See also

Game::update method and is used to update

fireTimer_

Reimplemented in PoisonTower, FreezingTower, and BombTower.

5.20.2.4 updateFireTimer()

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

Increments fireTimer_ by dt.

Parameters

dt	is time since last frame and is passed from

See also

Game::update()

5.20.2.5 upgradeTower()

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

upgradeTower() method is virtual as upgrade logic is same for all types of towers except Freezing Tower

Upgrade the tower to the next level.

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

Note

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

Reimplemented in FreezingTower.

5.20.3 Member Data Documentation

5.20.3.1 type_

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

Private members of abstract tower class.

Parameters

*type_*	is a string representing type of the tower
*baseCost_*	is the base cost for the type of tower
*range_*	is the enemy locking range of the tower
*damage_*	is passed as a parameter to projectile constructor
*currentLvl_*	is current level of the tower, initially set 1 and can be upgraded to level 2
*upgradeCost_*	is set at 1.5 $*$ base cost of tower for all types of towers
*lockedEnemy_*	is the locked enemy of the tower; initially set to nullptr

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

- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/tower.hpp
- /mnt/c/cpp-course-autumn-2023/tower-defense-tran-duong-2/src/tower.cpp