

Orcs n Towers

Generated by Doxygen 1.9.5

1 Source content	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 BombProjectile Class Reference	9
5.1.1 Detailed Description	9
5.1.2 Constructor & Destructor Documentation	9
5.1.2.1 BombProjectile()	9
5.1.3 Member Function Documentation	10
5.1.3.1 hasHitEnemy()	10
5.1.3.2 textureType()	10
5.1.3.3 update()	10
5.2 BombTower Class Reference	11
5.2.1 Member Function Documentation	11
5.2.1.1 getClassObject()	11
5.2.1.2 shoot()	11
5.2.1.3 update()	11
5.3 BulletProjectile Class Reference	12
5.3.1 Detailed Description	12
5.3.2 Member Function Documentation	12
5.3.2.1 hasHitEnemy()	12
5.3.2.2 textureType()	13
5.3.2.3 update()	13
5.4 BulletTower Class Reference	13
5.4.1 Member Function Documentation	13
5.4.1.1 getClassObject()	14
5.4.1.2 shoot()	14
5.5 Button Class Reference	14
5.6 Enemy Class Reference	14
5.7 Game Class Reference	15
5.8 LevelManager Class Reference	16
5.8.1 Constructor & Destructor Documentation	16
5.8.1.1 LevelManager()	16
5.8.2 Member Function Documentation	16
5.8.2.1 readingSuccessfull()	17
5.8.2.2 update()	17

5.9 Map Class Reference	17
5.10 Menu Class Reference	18
5.10.1 Detailed Description	18
5.10.2 Member Function Documentation	18
5.10.2.1 checkButtons()	18
5.10.2.2 createMenu()	19
5.10.2.3 drag()	19
5.10.2.4 draw()	19
5.10.2.5 update()	20
5.11 MissileProjectile Class Reference	20
5.11.1 Detailed Description	21
5.11.2 Constructor & Destructor Documentation	21
5.11.2.1 MissileProjectile()	21
5.11.3 Member Function Documentation	21
5.11.3.1 hasHitEnemy()	21
5.11.3.2 textureType()	21
5.11.3.3 update()	22
5.12 MissileTower Class Reference	22
5.12.1 Member Function Documentation	22
5.12.1.1 getClassObject()	22
5.12.1.2 shoot()	23
5.13 path Class Reference	23
5.14 Player Class Reference	23
5.14.1 Member Function Documentation	24
5.14.1.1 addMoney()	24
5.14.1.2 addToScore()	24
5.14.1.3 removeHP()	25
5.14.1.4 removeMoney()	25
5.15 Projectile Class Reference	25
5.15.1 Constructor & Destructor Documentation	26
5.15.1.1 Projectile()	26
5.15.2 Member Function Documentation	27
5.15.2.1 destroy()	27
5.15.2.2 hasHitEnemy()	27
5.15.2.3 textureType()	27
5.15.2.4 update()	27
5.16 ResourceContainer< T_enum, T_resource > Class Template Reference	28
5.17 Tower Class Reference	28
6 File Documentation	31
6.1 bombProjectile.hpp	31
6.2 bombTower.hpp	31

6.3 bulletProjectile.hpp	31
6.4 bulletTower.hpp	32
6.5 button.hpp	32
6.6 enemy.hpp	33
6.7 game.hpp	34
6.8 levelManager.hpp	35
6.9 map.hpp	36
6.10 menu.hpp	37
6.11 missileProjectile.hpp	37
6.12 missileTower.hpp	38
6.13 path.hpp	38
6.14 player.hpp	38
6.15 projectile.hpp	39
6.16 resource_container.hpp	40
6.17 tower.hpp	41
Index	43

Chapter 1

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 2

Hierarchical Index

2.1 Class Hierarchy

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

sf::Drawable	
Map	17
Game	15
LevelManager	16
Menu	18
path	23
ResourceContainer< T_enum, T_resource >	28
ResourceContainer< Textures::EnemyID, sf::Texture >	28
ResourceContainer< Textures::ProjectileID, sf::Texture >	28
ResourceContainer< Textures::TowerID, sf::Texture >	28
ResourceContainer< Textures::Various, sf::Texture >	28
sf::Sprite	
Button	14
Enemy	14
Player	23
Projectile	25
BombProjectile	9
BulletProjectile	12
MissileProjectile	20
Tower	28
BombTower	11
BulletTower	13
MissileTower	22
sf::Transformable	
Map	17

Chapter 3

Class Index

3.1 Class List

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

BombProjectile	
Projectile that causes damage to multiple enemies	9
BombTower	11
BulletProjectile	
Projectile that travels in a straight line and can hit only one enemy	12
BulletTower	13
Button	14
Enemy	14
Game	15
LevelManager	16
Map	17
Menu	
Class for storing a collection of buttons, a menu	18
MissileProjectile	
Projectile that targets (follows) a specific enemy	20
MissileTower	22
path	23
Player	23
Projectile	25
ResourceContainer< T_enum, T_resource >	28
Tower	28

Chapter 4

File Index

4.1 File List

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

/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ bombProjectile.hpp	31
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ bombTower.hpp	31
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ bulletProjectile.hpp	31
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ bulletTower.hpp	32
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ button.hpp	32
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ enemy.hpp	33
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ game.hpp	34
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ levelManager.hpp	35
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ map.hpp	36
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ menu.hpp	37
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ missileProjectile.hpp	37
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ missileTower.hpp	38
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ path.hpp	38
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ player.hpp	38
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ projectile.hpp	39
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ resource_container.hpp	40
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/ tower.hpp	41

Chapter 5

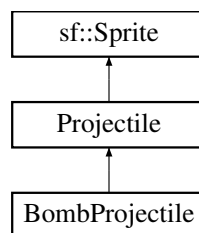
Class Documentation

5.1 BombProjectile Class Reference

a projectile that causes damage to multiple enemies

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

Inheritance diagram for BombProjectile:



Public Member Functions

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

returns the texture ID of the type this derived class uses

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 ) [inline]
```

Parameters

<i>blast</i> ↔ <i>Range_</i>	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

Implements [Projectile](#).

5.1.3.2 textureType()

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

returns the texture ID of the type this derived class uses

Implements [Projectile](#).

5.1.3.3 update()

```
void BombProjectile::update (
    Game & game ) [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

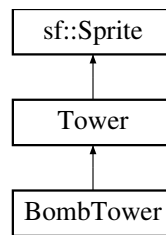
Implements [Projectile](#).

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bombProjectile.hpp
- /home/ottolitkey/cpp/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) override
- [BombProjectile](#) & **shoot** () override
- std::shared_ptr< [Tower](#) > **getClassObject** () override

Additional Inherited Members

5.2.1 Member Function Documentation

5.2.1.1 getClassObject()

```
std::shared_ptr< Tower > BombTower::getClassObject ( ) [override], [virtual]
```

Implements [Tower](#).

5.2.1.2 shoot()

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

Implements [Tower](#).

5.2.1.3 update()

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

Reimplemented from [Tower](#).

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

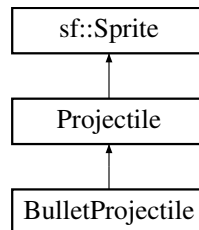
- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/bombTower.hpp
- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/bombTower.cpp

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:



Public Member Functions

- **BulletProjectile** (sf::Vector2f shootDirection, sf::Vector2f position, int damage)
- bool **hasHitEnemy** (std::shared_ptr< [Enemy](#) > &enemy)
- void **update** ([Game](#) &game)
- Textures::ProjectileID **textureType** ()
returns the texture ID of the type this derived class uses
- 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 ) [virtual]
```

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

Implements [Projectile](#).

5.3.2.2 textureType()

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

returns the texture ID of the type this derived class uses

Implements [Projectile](#).

5.3.2.3 update()

```
void BulletProjectile::update (
    Game & game ) [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 above has happened, the bullet is moved.

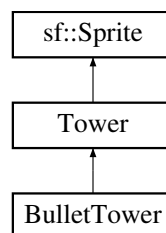
Implements [Projectile](#).

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletProjectile.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletProjectile.cpp

5.4 BulletTower Class Reference

Inheritance diagram for BulletTower:



Public Member Functions

- **BulletTower** (sf::Vector2f)
- [BulletProjectile](#) & [shoot](#) () override
- std::shared_ptr< [Tower](#) > [getClassObject](#) () override

Additional Inherited Members

5.4.1 Member Function Documentation

5.4.1.1 getClassObject()

```
std::shared_ptr< Tower > BulletTower::getClassObject ( ) [override], [virtual]
```

Implements [Tower](#).

5.4.1.2 shoot()

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

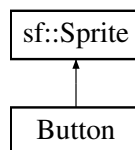
Implements [Tower](#).

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletTower.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletTower.cpp

5.5 Button Class Reference

Inheritance diagram for Button:



Public Member Functions

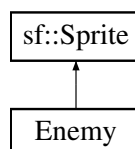
- **Button** (Actions action, sf::Texture &texture, sf::Vector2f position, std::string text, sf::Font &font)
- bool **isClicked** (sf::Vector2f mousePos) const
- Actions **getAction** () const
- sf::Text **getLabel** () const

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/button.hpp

5.6 Enemy Class Reference

Inheritance diagram for Enemy:



Public Member Functions

- **Enemy** (int hp, int speed, EnemyType type, int money, std::queue< sf::Vector2f > waypoints)
- void **update** (sf::Time time)
- sf::Vector2f **getCenter** ()
- sf::Vector2f **getLocation** ()
- bool **dead** ()
- int **hp** ()
- int **initialHp** ()
- float **speed** ()
- int **poisonStatus** ()
- int **slowedStatus** ()
- EnemyType **type** ()
- void **takeDamage** (int damage)
- void **kill** ()
- void **applyPoison** (int duration)
- void **poisonDamage** ()
- void **applySlowed** (int duration)
- void **slowedDamage** ()
- void **setVelocity** ()
- bool **isWaypointPassed** (sf::Vector2f movement)
- void **findNewWaypoint** ()
- std::queue< sf::Vector2f > **getWaypoints** ()
- void **moveEnemy** (sf::Vector2f movement)
- int **getMoney** () const

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

- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/enemy.hpp
- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/enemy.cpp

5.7 Game Class Reference

Public Member Functions

- void **run** ()

Public Attributes

- [Map](#) **map**

Friends

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

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

- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/game.hpp
- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/game.cpp

5.8 LevelManager Class Reference

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** ()
returns the current level
- int **getLevelTotal** ()
returns the total number of levels definend
- void [update](#) ()
- bool [readingSuccessful](#) ()

5.8.1 Constructor & Destructor Documentation

5.8.1.1 LevelManager()

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

Initialises a levelManager reads the level information from file

See also

[readLevels\(\)](#) initial current level is zero (= level one) to follow indexing convention of level specifications container to allow easier accessing

Parameters

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

5.8.2 Member Function Documentation

5.8.2.1 readingSuccessfull()

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

returns status flag for reading level info from file true if reading was successfull false if not

5.8.2.2 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 waitTime becomes zero, if there is waves left for the level Moves to a new level once previous is complete and there are no enemies left

See also

initiateEnemies()

Parameters

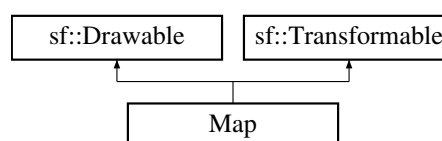
<i>game</i>	is a reference to the game instance that is responsible for calling update
-------------	--

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/levelManager.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/levelManager.cpp

5.9 Map Class Reference

Inheritance diagram for Map:



Public Member Functions

- void **loadMap** (const std::string &fileName)
- void **update** ()
- bool **canTowerBuild** (const std::shared_ptr< Tower > &activeTower) const
- void **buildTower** (const std::shared_ptr< Tower > &activeTower)
- void **sellTower** (Tower *sellingTower)
- sf::FloatRect **getBackgroundBounds** ()

Public Attributes

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

Friends

- class **MainGame**
- class **TowerManagement**

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/map.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/map.cpp

5.10 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.

5.10.1 Detailed Description

Class for storing a collection of buttons, a menu.

5.10.2 Member Function Documentation

5.10.2.1 checkButtons()

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

Checks if a button in the menu has been pressed.

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

Parameters

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

5.10.2.2 createMenu()

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

Creates the buttons and texts of a menu.

Parameters

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

5.10.2.3 drag()

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

Implements drag&drop placing of towers.

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

Parameters

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

See also

canBePlaced()

5.10.2.4 draw()

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

Draws all the objects in the menu.

Parameters

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

5.10.2.5 update()

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

Updates the status of the menu.

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

Parameters

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

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

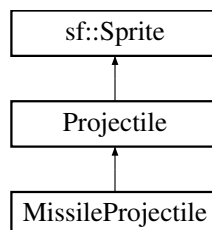
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/menu.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/menu.cpp

5.11 MissileProjectile Class Reference

a projectile that targets (follows) a specific enemy

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

Inheritance diagram for MissileProjectile:

**Public Member Functions**

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

5.11.1 Detailed Description

a projectile that targets (follows) a specific enemy

5.11.2 Constructor & Destructor Documentation

5.11.2.1 MissileProjectile()

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

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

Parameters

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

5.11.3 Member Function Documentation

5.11.3.1 hasHitEnemy()

```
bool MissileProjectile::hasHitEnemy (
    std::shared_ptr< Enemy > & enemy ) [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 and returns true.

Implements [Projectile](#).

5.11.3.2 textureType()

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

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

Implements [Projectile](#).

5.11.3.3 update()

```
void MissileProjectile::update (
    Game & game ) [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 it has hit it, if there's been a hit, the missile is destroyed. If it 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

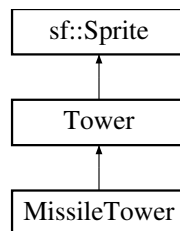
Implements [Projectile](#).

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileProjectile.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileProjectile.cpp

5.12 MissileTower Class Reference

Inheritance diagram for MissileTower:



Public Member Functions

- **MissileTower** (sf::Vector2f)
- [MissileProjectile](#) & [shoot](#) () override
- std::shared_ptr< [Tower](#) > [getClassObject](#) () override

Additional Inherited Members

5.12.1 Member Function Documentation

5.12.1.1 getClassObject()

```
std::shared_ptr< Tower > MissileTower::getClassObject ( ) [override], [virtual]
```

Implements [Tower](#).

5.12.1.2 shoot()

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

Implements [Tower](#).

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

- `/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileTower.hpp`
- `/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileTower.cpp`

5.13 path Class Reference

Public Member Functions

- 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**

Static Public Attributes

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

Friends

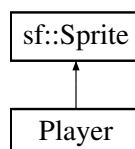
- class **enemy**

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

- `/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/path.hpp`
- `/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/path.cpp`

5.14 Player Class Reference

Inheritance diagram for Player:



Public Member Functions

- **Player** (std::string name="player")
- int **getWallet** () const
returns how much money the player has
- int **getHP** () const
returns how many health points the player has
- std::string **getName** () const
returns the name of the player
- int **getLevel** () const
returns the current level of the player
- 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 pointe from the player
- void **addToScore** (int amount)
adds points to the players score

5.14.1 Member Function Documentation

5.14.1.1 addMoney()

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

adds money to the players wallet

Parameters

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

5.14.1.2 addToScore()

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

adds points to the players score

Parameters

<i>amount</i>	is how many points is to be added
---------------	-----------------------------------

5.14.1.3 removeHP()

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

removes health pointe from the player

Parameters

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

5.14.1.4 removeMoney()

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

removes money from the players wallet

Parameters

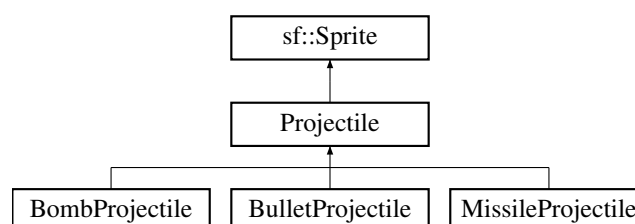
<i>cost</i>	is how much money is to be removed
-------------	------------------------------------

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/player.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/player.cpp

5.15 Projectile Class Reference

Inheritance diagram for Projectile:



Public Member Functions

- [Projectile](#) (sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed, std::string type, int maxDistance)
Constructs a projectile and sets it's position.
- float **getSpeed** () const
returns the speed of the projectile
- const std::string & **getType** () const
returns the type of the projectile
- int **getDamage** () const
returns the damage of the projectile
- sf::Vector2f **getShootDir** () const
returns the directional vector of the projectile
- void **destroy** ()
- bool **isDestroyed** ()
Returns wheter the projectile is destroyed, and need to be deleted, or not.
- bool **distToTower** ()
Calculates the distance from the tower that created it returns true if the projectile is at, or has exceeded, its maximum distance.
- virtual bool **hasHitEnemy** (std::shared_ptr< [Enemy](#) > &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

5.15.1 Constructor & Destructor Documentation

5.15.1.1 Projectile()

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

Constructs a projectile and sets it's position.

Parameters

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

5.15.2 Member Function Documentation

5.15.2.1 destroy()

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

Sets the

Parameters

<i>isDestroyed</i>	flag to true when the projectile has hit an enemy, and fulfilled its purpose, or when it has gone out of range (exceeded its max distance), and needs to be destroyed
—	

5.15.2.2 hasHitEnemy()

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

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

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

5.15.2.3 textureType()

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

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

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

5.15.2.4 update()

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

updates the projectiles state as is defined in each derived class

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

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

- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/projectile.hpp
- /home/ottolilkey/cpp/tower-defense-tran-duong-2/src/projectile.cpp

5.16 ResourceContainer< T_enum, T_resource > Class Template Reference

Public Member Functions

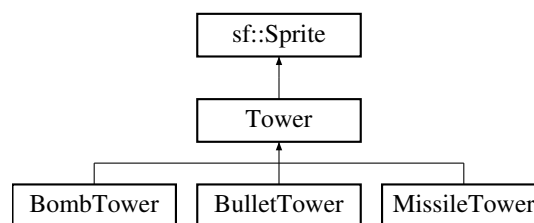
- void **load** (T_enum type, std::string filename)
- T_resource & **get** (T_enum type) const

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/resource_container.hpp

5.17 Tower Class Reference

Inheritance diagram for Tower:



Public Member Functions

- **Tower** (sf::Vector2f position, const std::string &type="Basic", int baseCost=100, float range=100.0, float fireRate=1.0, int damage=10, int currentLvl=1, int upgradeCost=150, CanDamage damageType=CanDamage::Both, std::shared_ptr< [Enemy](#) > lockedEnemy=nullptr, sf::Clock fireTimer=sf::Clock(), bool maxLevelReached=false)
- const std::string & **getType** () const
- const int **getBaseCost** () const
- const float **getFireRate** () const
- const CanDamage **getDamageType** () const
- const float **getRange** () const
- int **getDamage** () const
- std::shared_ptr< [Enemy](#) > **getLockedEnemy** () const
- void **setLockedEnemy** (std::shared_ptr< [Enemy](#) > enemy)
- bool **isMaxLevelReached** () const
- int **getCurrentLvl** () const
- const int **getUpgradeCost** () const
- sf::Clock **getFireTimer** ()
- bool **enemyWithinRange** (std::shared_ptr< [Enemy](#) > enemy)
- void **resetFireTimer** ()
- virtual [Projectile](#) & **shoot** ()=0
- void **upgradeTower** ()
- virtual void **update** (std::list< std::shared_ptr< [Enemy](#) > > &enemies)
- virtual std::shared_ptr< [Tower](#) > **getClassObject** ()=0
- bool **isActive** ()
- void **unactiveHUD** ()
- void **activateHUD** ()
- sf::Vector2f **getSize** ()
- virtual void **build** ()

Public Attributes

- bool **HUDactive** = false
- bool **buildid** = false

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

- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/tower.hpp
- /home/ottolitkey/cpp/tower-defense-tran-duong-2/src/tower.cpp

Chapter 6

File Documentation

6.1 bombProjectile.hpp

```
1 #ifndef BOMB_PROJECTILE
2 #define BOMB_PROJECTILE
3
4 #include "projectile.hpp"
5 #include <list>
6
10 class BombProjectile : public Projectile
11 {
12 private:
13     int blastRange_;
14 public:
15
19     BombProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage) // <- tbd
20     : Projectile(shootDirection, position, damage, 60.0, "bomb", 50), blastRange_(1000) {}
21     // ^ ^ ^ tbd
22
27     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) override;
28
34     void update(Game& game);
35
39     Textures::ProjectileID textureType() { return Textures::Bomb; }
40 };
41
42
43 #endif
```

6.2 bombTower.hpp

```
1 #ifndef BOMB_TOWER_H
2 #define BOMB_TOWER_H
3 #include "tower.hpp"
4 #include "bombProjectile.hpp"
5 class BombTower : public Tower {
6 public:
7     BombTower(sf::Vector2f);
8     void update(std::list<std::shared_ptr<Enemy>> &enemies) override;
9     BombProjectile& shoot() override;
10     std::shared_ptr<Tower> getClassObject() override;
11 private:
12     // void draw() override;
13     // Draw method is inherited from sf::Sprite so I have commented this out for now
14     // If we want to include some animation stuff I'm not sure where it would be best to do
15 };
16 #endif //BOMB_TOWER_H
```

6.3 bulletProjectile.hpp

```
1 #ifndef BULLET_PROJECTILE
2 #define BULLET_PROJECTILE
```

```

3
4 #include "projectile.hpp"
5
6 class BulletProjectile : public Projectile
7 {
8 public:
9     BulletProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage)
10     : Projectile(shootDirection, position, damage, 300, "bullet", 70) {}
11
12     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy);
13
14     void update(Game& game);
15
16     Textures::ProjectileID textureType() { return Textures::Bullet; }
17
18     float rotationAngle() const; //this one is used to calculate rotation angle of a projectile.
19 };
20
21 #endif

```

6.4 bulletTower.hpp

```

1 #ifndef BULLET_TOWER_H
2 #define BULLET_TOWER_H
3 #include "tower.hpp"
4 #include "bulletProjectile.hpp"
5 class BulletTower : public Tower {
6 public:
7     BulletTower(sf::Vector2f);
8     /*update() method for BulletTower uses update() from base class,
9     hence I have commented it out*/
10    //void update(std::list<std::shared_ptr<Enemy>> &enemies) override;
11    BulletProjectile& shoot() override;
12    std::shared_ptr<Tower> getClassObject() override;
13 private:
14    //void draw() override;
15 };
16 #endif //BULLET_TOWER_H

```

6.5 button.hpp

```

1 #ifndef BUTTON
2 #define BUTTON
3 #include <SFML/Graphics.hpp>
4
5 enum class Actions{
6     Tower1,
7     Tower2,
8     Tower3,
9     Tower4,
10    Tower5,
11    Pause,
12    Upgrade,
13    Sell,
14    Close, // In upgrade menu, closes upgrade menu.
15    Level // Click to start level
16 };
17
18 class Button : public sf::Sprite {
19 public:
20     Button(Actions action, sf::Texture& texture, sf::Vector2f position, std::string text, sf::Font& font)
21     : action_(action) {
22         setTexture(texture);
23         setPosition(position);
24         label_ = sf::Text(text, font, 15);
25         label_.setPosition(position.x, position.y+20);
26     }
27     bool isClicked(sf::Vector2f mousePos) const {
28         return getGlobalBounds().contains(mousePos);
29     }
30
31     Actions getAction() const {return action_;}
32     sf::Text getLabel() const {return label_;}
33 private:
34     Actions action_;
35     sf::Text label_;
36 }

```

```

37 };
38
39
40 #endif

```

6.6 enemy.hpp

```

1 #ifndef ENEMY_HPP
2 #define ENEMY_HPP
3 #include <string>
4 #include "path.hpp"
5 #include <queue>
6 #include "player.hpp"
7 #include <SFML/System/Vector2.hpp>
8 #include <SFML/Graphics.hpp>
9
10 enum class EnemyType {
11     Ground,
12     Flying,
13     Split,
14 };
15
16 class Enemy :public sf::Sprite {
17 public:
18
19     Enemy(int hp, int speed, EnemyType type, int money, std::queue<sf::Vector2f> waypoints)
20         : hp_(hp), actualSpeed_(speed), speed_(speed), effectiveSpeed_(speed), type_(type),
21         money_(money), waypoints_(waypoints) {
22         //Ellen: enemies can set their own position, since it will be same for everyone
23         setPosition(waypoints_.front());
24
25         if (!waypoints_.empty()) {
26             currentWaypoint_ = waypoints_.front();
27         }
28         setVelocity();
29     }
30
31
32     ~Enemy() {}
33
34     void update(sf::Time time); //update the state of the monster in relation to the game
35
36     sf::Vector2f getCenter();
37
38     sf::Vector2f getLocation();
39
40     bool dead();
41
42     int hp();
43
44     int initialHp();
45
46     float speed();
47
48     int poisonStatus();
49
50     int slowedStatus();
51
52     EnemyType type();
53
54     void takeDamage(int damage); //decreases the hp_ variable and if hp reaches 0 than the enemy is
55     automatically destroyed
56
57     void kill();
58
59     void applyPoison(int duration);
60
61     void poisonDamage();
62
63     void applySlowed(int duration);
64
65     void slowedDamage();
66
67     void setVelocity();
68
69     bool isWaypointPassed(sf::Vector2f movement);
70
71     void findNewWaypoint();
72
73     std::queue<sf::Vector2f> getWaypoints();
74
75     void moveEnemy(sf::Vector2f movement);

```

```

75
76     int getMoney() const;
77
78
79 private:
80     int hp_;
81
82     int initialHp_;
83
84     bool dead_= false;
85
86     float speed_;
87
88     float actualSpeed_;
89
90     float effectiveSpeed_;
91
92     EnemyType type_;
93     //has reference to player instance so money can be deposited to the player as well as the use of
    other player functions
94
95     int poison_=0; //If poison is larger than 0 that means that the enemy is poisoned
96     // the length of time that the enemy is poisoned for depends on how large the poison
97     //value is as the number decreases incremently until 0
98     int slowed_=0;
99     //How much money the player recieves for killing the monster
100     int money_;
101     //waypoint based movement, the path class provides a queue of waypoints that take the enemies
    through the path to the end
102
103     sf::Vector2f velocity_;
104
105     std::queue<sf::Vector2f> waypoints_;
106
107     sf::Vector2f currentWaypoint_;
108
109     int direction_; //0 = down, 1= left, 2= right, 3 = up
110
111 };
112
113 #endif

```

6.7 game.hpp

```

1 #ifndef GAME_HPP
2 #define GAME_HPP
3
4 #include <SFML/Graphics.hpp>
5 #include <list>
6 #include "tower.hpp"
7 #include "path.hpp"
8 #include "enemy.hpp"
9 #include "projectile.hpp"
10 #include "resource_container.hpp"
11 #include "player.hpp"
12 #include <memory> //for shared_ptr
13 #include "bulletTower.hpp"
14 #include "button.hpp"
15 #include "map.hpp"
16 #include "missileProjectile.hpp"
17 #include "menu.hpp"
18 #include <vector>
19 #include "levelManager.hpp"
20
21 class Menu;
22 // Class for running the game logic
23
24 class Game {
25
26     friend class Tower;
27     friend class BulletTower;
28     friend class BombTower;
29     friend class MissileTower;
30     friend class BombProjectile;
31     friend class BulletProjectile;
32     friend class MissileProjectile;
33     friend class Menu;
34     friend class LevelManager;
35
36 public:
37     Map map;
38     Game();
39     void run();

```



```

40
41 ~Game() {
42
43     for(auto i : enemies_){
44         //delete i;
45     }
46     enemies_.clear();
47
48     for(auto i : projectiles_){
49         delete i;
50     }
51     projectiles_.clear();
52
53     for(auto i : towers_){
54         delete i;
55     }
56     towers_.clear();
57 }
58
59 private:
60     void processEvents();
61     void update();
62     void render();
63     void addTower(const Tower& tower);
64     void addEnemy(const Enemy& enemy);
65     void addProjectile(const Projectile& projectile);
66     void createPath(); //this will create the path that the enemies will traverse (this should also be
        rendered visually in the game)
67     void checkTowers();
68     void testEnemy();
69     void testEnemySplit(sf::Vector2f position, std::queue<sf::Vector2f> waypoints);
70     //adding a function to return the elapsed time
71     sf::Time getTime() const;
72     //I am adding a clock and time functionality that will need to be used for enemy movement and
        updating and other game logic
73     sf::Clock clock_;
74     sf::Time time_;
75     sf::RenderWindow window_;
76     /* Pavel: should we change enemies_ to be
77     * std::list<std::shared_ptr<Enemy> enemies_ instead and
78     * initialize it with enemies using
79     * enemies_.push_back(std::make_shared<TYPE_OF_ENEMY>(args))?
80     * This way enemy would get destroyed automatically when
81     * it is no longer locked by any tower AND it has reached 0 hp
82     * (it gets removed from the list at this point) AND no projectile
83     * flies towards it (we need to add shared_ptr<Enemy> member to projectile class).
84     * There might be some error in my logic though....
85     */
86     std::list<Tower*> towers_;
87     //std::list<Enemy> enemies_;
88
89     /* Changed these to unque ptr, as looping over the abstract types directly
90     is not possible, at least according to my understanding
91     */
92     std::list<std::shared_ptr<Enemy> enemies_;
93     std::list<Projectile*> projectiles_;
94     path path_;
95     std::list<Button> buttons_; // Stores clickable buttons
96     bool dragged_; // Indicates if a tower is currently being dragged into place
97     bool paused_; // Is the game paused?
98     bool isGameOver_ = false; //is the game over because the player has died to an enemy
99     sf::Font font_; // Stores text font
100     sf::Text gameOverText;
101     sf::Sprite castle_sprite;
102
103     Menu* shop_; // Shop on left side
104     Menu* alternativeMenu_; // stores menu for upgrading, beginning game, and advancing to next level
105     Tower* activeTower_; // Pointer to tower that is being upgraded
106
107     ResourceContainer<Textures::TowerID, sf::Texture> tower_textures_;
108     ResourceContainer<Textures::EnemyID, sf::Texture> enemy_textures_;
109     ResourceContainer<Textures::ProjectileID, sf::Texture> projectile_textures_;
110     ResourceContainer<Textures::Various, sf::Texture> various_textures_;
111
112     Player player_;
113
114     LevelManager levelManager_;
115 };
116
117 #endif

```

6.8 levelManager.hpp

```
1 #ifndef LEVELMANAGER
```

```

2 #define LEVELMANAGER
3
4 #pragma once
5
6 #include <iostream>
7 #include <string>
8 #include <vector>
9 #include <map>
10 #include <variant>
11 #include <fstream>
12 #include <sstream>
13 #include <random>
14
15 #include "enemy.hpp"
16 #include "path.hpp"
17
18 class Game;
19
20 class LevelManager {
21
22     public:
23         //this requires c++17
24         using variantData = std::variant<int, float, std::vector<int>>;
25
26         LevelManager(const std::string& src, path& path, Game& game, Player& player) : src_(src),
27         path_(path), game_(game), player_(player) {
28             readLevels();
29
30             currLevel_ = 0;
31             waitTime_ = 0;
32             levelTotal_ = levelSpecs_.size();
33         }
34         ~LevelManager(){}
35
36         int getCurrentLevel();
37
38         int getLevelTotal();
39
40         void update();
41
42         bool readingSuccessfull(); //need to abort game if reading fails, use this to check
43
44         //void printLevelSpecs();
45
46     private:
47         void readLevels();
48
49         void initiateEnemies();
50
51         std::vector<std::map<std::string, variantData>> levelSpecs_;
52
53         int currLevel_;
54         const std::string& src_;
55         bool readingSuccess_;
56         int levelTotal_;
57         float waitTime_;
58
59         path& path_;
60         Game& game_;
61         Player& player_;
62 };
63
64 #endif

```

6.9 map.hpp

```

1 #ifndef MAP_HPP
2 #define MAP_HPP
3
4 #include <SFML/Graphics.hpp>
5 #include <memory>
6 #include <string>
7 #include <vector>
8 #include "tower.hpp"
9
10 class Tower; // Forward declaration
11
12 class Map : public sf::Drawable, public sf::Transformable {
13     friend class MainGame;
14     friend class TowerManagement;
15

```

```

16
17
18 public:
19     sf::Texture texture;
20     sf::Sprite background;
21     Map();
22     ~Map();
23
24     void loadMap(const std::string& fileName);
25     void update();
26     bool canTowerBuild(const std::shared_ptr<Tower>& activeTower) const;
27     void buildTower(const std::shared_ptr<Tower>& activeTower);
28     void sellTower(Tower* sellingTower);
29     std::vector<sf::FloatRect> unBuildable;
30
31     sf::FloatRect getBackgroundBounds();
32
33 private:
34     std::vector<std::shared_ptr<Tower>> towers;
35
36
37     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
38 };
39
40 #endif // MAP_HPP

```

6.10 menu.hpp

```

1 #ifndef MENU
2 #define MENU
3 #include <SFML/Graphics.hpp>
4 #include <list>
5 #include "button.hpp"
6 #include "game.hpp"
7 #include "tower.hpp"
8
9 // These are used in createMenu()
10 // the enum determines what type of menu is created:
11 // Which buttons are added etc.
12 enum class MenuType{
13     Shop,
14     Upgrade,
15     Begin,
16     Level
17 };
18
19 class Menu {
20 public:
21     void draw(sf::RenderWindow& window);
22
23     void checkButtons(Game* game);
24
25     void createMenu(MenuType menu, Game* game);
26
27     void update(Player& player);
28
29     void drag(Game* game);
30 private:
31     void newTower(Tower* tower, Game* game);
32
33     bool canBePlaced(Game* game);
34
35     std::list<Button> buttons_;
36     std::list<sf::Text> texts_;
37     sf::RectangleShape bg_;
38 };
39
40 #endif

```

6.11 missileProjectile.hpp

```

1 #ifndef MISSILE_PROJECTILE
2 #define MISSILE_PROJECTILE
3
4 #include "projectile.hpp"
5
6 class MissileProjectile : public Projectile
7 {
8 private:

```

```

12     std::shared_ptr<Enemy> targetEnemy_;
13
14 public:
21     MissileProjectile(sf::Vector2f position, int damage, std::shared_ptr<Enemy> targetEnemy)
22     : Projectile(sf::Vector2f(0,0), position, damage, 70.0, "missile", 70), targetEnemy_(targetEnemy) {}
23
29     bool hasHitEnemy(std::shared_ptr<Enemy>& enemy);
30
38     void update(Game& game);
39     Textures::ProjectileID textureType() { return Textures::Missile; }
40 };
41
42
43 #endif

```

6.12 missileTower.hpp

```

1 #ifndef MISSILE_TOWER
2 #define MISSILE_TOWER
3 #include "tower.hpp"
4 #include "missileProjectile.hpp"
5 class MissileTower : public Tower {
6 public:
7     MissileTower(sf::Vector2f);
8     MissileProjectile& shoot() override;
9     std::shared_ptr<Tower> getClassObject() override;
10 };
11 #endif

```

6.13 path.hpp

```

1 #ifndef PATH_HPP
2 #define PATH_HPP
3 #include <queue>
4 #include <SFML/System/Vector2.hpp>
5 #include <SFML/Graphics.hpp>
6 #include <vector>
7
8 class path {
9     friend class enemy;
10 public:
11     path(); //creates a path and populates the waypoints queue with
12     //all the waypoints required for the enemy class to traverse the path
13
14
15     ~path() {
16
17     }
18
19     void addWaypoint(const sf::Vector2f& point);
20
21     std::queue<sf::Vector2f> getWaypoints() const;
22     void makeUnBuildablePath();
23     static const float width;
24     std::queue<sf::Vector2f> waypoints_;
25     std::vector <sf::Vector2f> wayPoints;
26     std::vector <sf::FloatRect> unBuildable;
27
28 private:
29
30 };
31
32 #endif

```

6.14 player.hpp

```

1 #ifndef PLAYER
2 #define PLAYER
3
4 #include <string>
5 #include <list>
6 #include "enemy.hpp"
7 #include "tower.hpp"
8 #include <SFML/System/Vector2.hpp>
9 #include <SFML/Graphics/Transformable.hpp>

```

```

10 #include <memory>
11 #include "resource_container.hpp"
12
13 class Tower;
14 class Enemy;
15
19 class Player : public sf::Sprite
20 {
21     private:
22         int hp_;
23         int wallet_;
24         std::string name_;
25         int score_;
26         sf::Vector2f position_; //of castle
27         int level_;
28
29     public:
30         Player(std::string name = "player")
31             // position_ cannot be initialized as NULL, because it is of type sf::Vector2f
32             :name_(name), hp_(500), wallet_(500), score_(0), position_(sf::Vector2f(0, 0)), level_(0){}
33
34         ~Player() {}
35
39         int getWallet() const;
40
44         int getHP() const;
45
49         std::string getName() const;
50
54         int getLevel() const;
55
59         void levelUp();
60
65         void addMoney(int amount);
66
71         void removeMoney(int cost);
72
77         void removeHP(int amount);
78
83         void addToScore(int amount);
84
85         //void reachedCastle (std::shared_ptr<Enemy>& enemy); //checks if an enemy has reached the castle
or should game do it?
86         //std::list<std::shared_ptr<Enemy>> increaseLevel(ResourceContainer<Textures::EnemyID,
sf::Texture>& enemytextures, path& path);
87         //void buyTower(int cost, Textures::TowerID towerID);
88         //void sellTower(Tower& tower);
89         //void upgradeTower(Tower& tower);
90
96         //void updateCastlePosition(sf::Vector2f position);
97 };
98
99 #endif

```

6.15 projectile.hpp

```

1 #ifndef PROJECTILE
2 #define PROJECTILE
3
4 #include "tower.hpp"
5 #include "player.hpp"
6 #include "enemy.hpp"
7 #include "resource_container.hpp"
8 #include <SFML/System/Vector2.hpp>
9 #include <SFML/Graphics/Transformable.hpp>
10 #include <SFML/Graphics.hpp>
11 #include <memory>
12 #include <iostream>
13
14 class Game;
15 class Enemy;
16
17 // Removed inheritance of sf::Transformable.
18 // I checked that sf::Sprite inherits both Drawable and Transformable -Otto
19 class Projectile : public sf::Sprite
20 {
21     private:
22         float speed_;
23         std::string type_;
24         int damage_;
25         sf::Vector2f position_; // of tower that created
26         int maxDistance_;
27         sf::Vector2f shootDirection_;

```

```

28         bool isDestroyed_;
29
30     public:
31
41         Projectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed,
std::string type, int maxDistance)
42             : shootDirection_(shootDirection), position_(position), damage_(damage), speed_(speed),
type_(type), maxDistance_(maxDistance),
43             isDestroyed_(false){
44             this->setPosition(position_);
45         }
46
47         // shootdirection, position, damage comes from tower
48         // speed, type, maxDistance come from derived classes
49         // shootDirection needs to be normalised vector
50
51         virtual ~Projectile() {}
52
56         float getSpeed() const;
57
61         const std::string& getType() const;
62
66         int getDamage() const;
67
71         sf::Vector2f getShootDir() const;
72         //sf::Vector2f getVelocity() const;
73
78         void destroy();
79
83         bool isDestroyed();
84
89         bool distToTower();
90
95         virtual bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) = 0;
96
100        virtual void update(Game&) = 0;
101
106        virtual Textures::ProjectileID textureType() = 0;
107 };
108 #endif
109

```

6.16 resource_container.hpp

```

1  #ifndef RESOURCE_CONTAINER
2  #define RESOURCE_CONTAINER
3  #include <SFML/Graphics.hpp>
4  #include <string>
5  #include <memory>
6
7  // Enums for different textures
8  namespace Textures{
9
10     // NOTE: these could also be stored in one big enum...
11     enum TowerID {BulletTower, BombTower, MissileTower, Tower4, Tower5};
12     enum EnemyID {Enemy1, Enemy2, Enemy3, Enemy4, Enemy5};
13     enum ProjectileID{Bullet, Bomb, Missile};
14     enum Various {Pause, Castle};
15 }
16
17 // template class for loading and storing textures and other resources
18
19 template <typename T_enum, typename T_resource>
20 class ResourceContainer {
21 public:
22
23     // Load resource into container
24     void load(T_enum type, std::string filename){
25         std::unique_ptr<T_resource> resource(new T_resource());
26
27         if (!resource->loadFromFile(filename)){
28             //TODO: Handle texture loading error
29         }
30         // The function move should avoid creating a copy of the object resource, when inserting it into
the map
31         resources_.insert(std::make_pair(type, std::move(resource)));
32     }
33
34     // Find wanted resource, return reference
35     T_resource& get(T_enum type) const {
36         auto wanted = resources_.find(type);
37         return *wanted->second;
38     }
39

```

```

39
40
41 private:
42     std::map<T_enum, std::unique_ptr<T_resource>> resources_;
43
44
45 };
46
47 #endif

```

6.17 tower.hpp

```

1 #ifndef TOWER_H
2 #define TOWER_H
3 // EDIT: I think we should reduce max level of tower to lvl 2.
4 #define TOWER_MAX_LVL 2
5 #include <string>
6 #include <array>
7 #include <SFML/System/Vector2.hpp>
8 #include <SFML/System/Clock.hpp>
9 #include <SFML/Graphics.hpp>
10 #include "projectile.hpp"
11 #include "enemy.hpp"
12 #include <memory>
13
14 // enum class CanDamage is needed for implementing enemy-locking logic
15 // (i.e., which EnemyType can be locked and damaged by a specific type of tower).
16 class Projectile;
17
18 enum class CanDamage {
19     Ground,
20     Flying,
21     Both
22 };
23 /* Base tower class will be abstract (i.e., no objects of base tower class are to be constructable)*/
24 class Tower : public sf::Sprite {
25 public:
26     Tower(sf::Vector2f position, const std::string& type = "Basic", int baseCost = 100, float range =
100.0, float fireRate = 1.0,
27         int damage = 10, int currentLvl = 1, int upgradeCost = 150, CanDamage damageType =
CanDamage::Both, std::shared_ptr<Enemy> lockedEnemy = nullptr,
28         sf::Clock fireTimer = sf::Clock(), bool maxLevelReached = false);
29     /*Tower(sf::Vector2f position);*/
30     // I think there is really no need for copy constructor or copy assignment operator
31     const std::string& getType() const {return type_;}
32     //const sf::Vector2f getPosition() const {return position_;}
33     const int getBaseCost() const {return baseCost_;}
34     const float getFireRate() const {return fireRate_;}
35     const CanDamage getDamageType() const {return damageType_;}
36     const float getRange() const {return range_;}
37     int getDamage() const {return damage_;}
38     // const CanDamage getDamageType() const {return damageType_;}
39     std::shared_ptr<Enemy> getLockedEnemy() const {return lockedEnemy_;}
40     void setLockedEnemy(std::shared_ptr<Enemy> enemy) {lockedEnemy_ = enemy;}
41     bool isMaxLevelReached() const {return maxLevelReached_;}
42     int getCurrentLvl() const {return currentLvl_;}
43     const int getUpgradeCost() const {return upgradeCost_;}
44     sf::Clock getFireTimer() {return fireTimer_;}
45     bool enemyWithinRange(std::shared_ptr<Enemy> enemy);
46     void resetFireTimer() {fireTimer_.restart();}
47     // shoot() creates a projectile that flies towards lockedEnemy_
48     // Changed it to pure virtual
49     virtual Projectile& shoot() = 0;
50     void upgradeTower(); // Will be defined in .cpp
51     /* update() method is declared as virtual. Some derived
52     classes will use base update() and other will use override*/
53     virtual void update(std::list<std::shared_ptr<Enemy>> &enemies);
54     //This is what I add to support for the map class
55     virtual std::shared_ptr<Tower> getClassObject() = 0; //Type of Tower
56     bool isActive(); //Whether the Tower is active or not
57     bool HUDActive = false; //Temporary variable hold the state of the Tower
58     void unactiveHUD(); //Function to deactivate Tower (when sell)
59     void activateHUD(); //Function to activate Tower (when buy)
60     sf::Vector2f getSize(); //Get the height and width of the Tower we want to build
61     virtual void build(); //build function which change the temporary variable build to true
62     bool builded = false; //Temporary variable define whether the tower is built or not.
63 private:
64     // virtual void draw();
65     const std::string type_;
66     //const sf::Vector2f position_;
67     const int baseCost_;
68     const float range_;
69     int damage_;

```

```
70     const float fireRate_; // Rate at which tower creates new projectiles; perhaps fireRate shouldn't be
    upgradable, instead stronger projectiles are created
71     int currentLvl_;
72     const CanDamage damageType_;
73     const int upgradeCost_;
74     std::shared_ptr<Enemy> lockedEnemy_;
75     sf::Clock fireTimer_;
76     bool maxLevelReached_;
77 };
78 #endif //TOWER_H
```


Index

/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bombProjectile.h, 12
31
textureType, 12
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bombTower.cpp, 13
31
BulletTower, 13
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletProjectile.h, 13
31
ClassObject, 13
shoot, 14
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/bulletTower.h, 13
32
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/button.cpp, 18
32
checkButtons
Menu, 18
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/enemy.h, 19
33
createMenu
Menu, 19
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/game.h, 19
34
destroy
Projectile, 27
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/levelManager.h, 16
35
drag
Menu, 19
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/map.h, 19
36
draw
Menu, 19
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/menu.h, 19
37
Enemy, 14
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileProjectile.h, 20
37
Game, 15
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/missileTower.h, 22
38
getClassObject
BombTower, 11
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/path.h, 13
38
BulletTower, 13
MissileTower, 22
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/player.h, 24
38
hasHitEnemy
BombProjectile, 10
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/projectile.h, 10
39
BulletProjectile, 12
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/resource_container.h, 21
40
Projectile, 27
/home/ottolitkey/cpp/tower-defense-tran-duong-2/src/tower.h, 11
41
LevelManager, 16
LevelManager, 16
readingSuccessfull, 16
update, 17
addMoney
Player, 24
addToScore
Player, 24
BombProjectile, 9
BombProjectile, 9
hasHitEnemy, 10
textureType, 10
update, 10
BombTower, 11
getClassObject, 11
shoot, 11
update, 11
BulletProjectile, 12
Map, 17
Menu, 18
checkButtons, 18
createMenu, 19
drag, 19
draw, 19
update, 20
MissileProjectile, 20
hasHitEnemy, 21
MissileProjectile, 21
textureType, 21
update, 21

- MissileTower, [22](#)
 - getClassObject, [22](#)
 - shoot, [22](#)
- path, [23](#)
- Player, [23](#)
 - addMoney, [24](#)
 - addToScore, [24](#)
 - removeHP, [25](#)
 - removeMoney, [25](#)
- Projectile, [25](#)
 - destroy, [27](#)
 - hasHitEnemy, [27](#)
 - Projectile, [26](#)
 - textureType, [27](#)
 - update, [27](#)
- readingSuccessfull
 - LevelManager, [16](#)
- removeHP
 - Player, [25](#)
- removeMoney
 - Player, [25](#)
- ResourceContainer< T_enum, T_resource >, [28](#)
- shoot
 - BombTower, [11](#)
 - BulletTower, [14](#)
 - MissileTower, [22](#)
- textureType
 - BombProjectile, [10](#)
 - BulletProjectile, [12](#)
 - MissileProjectile, [21](#)
 - Projectile, [27](#)
- Tower, [28](#)
- update
 - BombProjectile, [10](#)
 - BombTower, [11](#)
 - BulletProjectile, [13](#)
 - LevelManager, [17](#)
 - Menu, [20](#)
 - MissileProjectile, [21](#)
 - Projectile, [27](#)