Orcs n Towers

Generated by Doxygen 1.9.8

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. 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 effect on the monster.

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.

1.2 Instructions

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

1.3 How to compile the program

To compile the game from 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 (minimum version 2.5) is required.

1.4 Testing

1.5 Work log

Division of work / main responisbilities:

Pavel:

- Tower class and it's derived classes
- Game class

Otto:

- Graphics
- Menu class

Ellen:

- Projectile class and it's derived classes
- LevelManager class
- Reading paths from file
- Player class

Leo:

- Enemy class

Tuan:

- 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

File Index

5.1 File List

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

Chapter 6

Class Documentation

6.1 BombProjectile Class Reference

a projectile that causes damage to multiple enemies

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

Inheritance diagram for BombProjectile:

class_bomb_projectile-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)
- Textures::ProjectileID textureType ()

  returns the texture ID of the type this derived class uses

Public Member Functions inherited from Projectile

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

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

  Destroy the Projectile object.
- 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 needs 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.

Private Attributes

- int **blastRange_**

6.1.1 Detailed Description

a projectile that causes damage to multiple enemies

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BombProjectile()

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

Parameters

*blast↩ Range_*	the blast radius of the bomb

6.1.3 Member Function Documentation

6.1.3.1 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

Parameters

enemy	is a reference to an Enemy object

Implements Projectile.

6.1.3.2 textureType()

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

returns the texture ID of the type this derived class uses

Implements Projectile.

6.1.3.3 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.

Parameters

game	is a reference to the running game instance

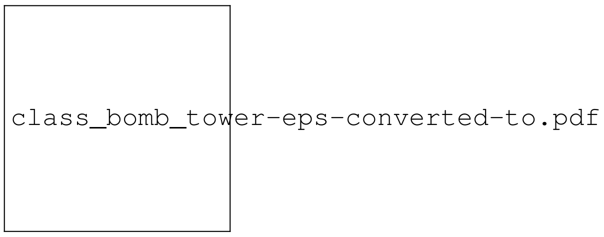
Implements Projectile.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bombProjectile.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bombProjectile.cpp

6.2 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

Public Member Functions inherited from Tower

- **Tower** (sf::Vector2f position, const std::string &type="Basic", int baseCost=100, float range=100.0, float fire↩ Rate=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 maxLevel↩ Reached=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** ()
- void **upgradeTower** ()
- bool **isActive** ()
- void **unactiveHUD** ()
- void **activateHUD** ()
- sf::Vector2f **getSize** ()
- virtual void **build** ()

Additional Inherited Members

Public Attributes inherited from Tower

- bool **HUDactive** = false
- bool **builded** = false

6.2.1 Member Function Documentation

6.2.1.1 getClassObject()

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

Implements Tower.

6.2.1.2 shoot()

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

Implements Tower.

6.2.1.3 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:

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bombTower.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bombTower.cpp

6.3 BulletProjectile Class Reference

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

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

Inheritance diagram for BulletProjectile:

class_bullet_projectile-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)

  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.
- 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.

Public Member Functions inherited from Projectile

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

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

  Destroy the Projectile object.
- 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 needs 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.

6.3.1 Detailed Description

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

6.3.2 Member Function Documentation

6.3.2.1 hasHitEnemy()

```
bool BulletProjectile::hasHitEnemy (
            std::shared_ptr< Enemy > & enemy )  [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.

Parameters

enemy	is a reference to an Enemy object

Implements Projectile.

6.3.2.2 textureType()

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

returns the texture ID of the type this derived class uses

Implements Projectile.

6.3.2.3 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.

Parameters

game	is a reference to the running game instance

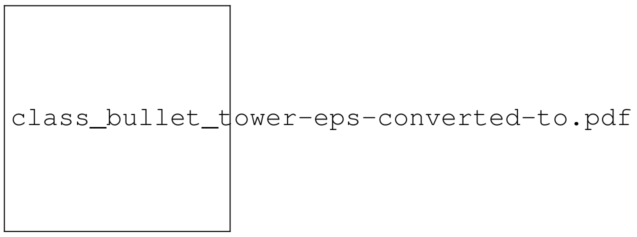
Implements Projectile.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bulletProjectile.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bulletProjectile.cpp

6.4 BulletTower Class Reference

Inheritance diagram for BulletTower:



Public Member Functions

- **BulletTower** (sf::Vector2f)
- BulletProjectile ∗ shoot () override
- std::shared_ptr< Tower > getClassObject () override

Public Member Functions inherited from Tower

- **Tower** (sf::Vector2f position, const std::string &type="Basic", int baseCost=100, float range=100.0, float fire↩ Rate=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 maxLevel↩ Reached=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** ()
- void **upgradeTower** ()
- virtual void **update** (std::list< std::shared_ptr< Enemy > > &enemies)
- bool **isActive** ()
- void **unactiveHUD** ()
- void **activateHUD** ()
- sf::Vector2f **getSize** ()
- virtual void **build** ()

Additional Inherited Members

Public Attributes inherited from Tower

- bool **HUDactive** = false
- bool **builded** = false

6.4.1 Member Function Documentation

6.4.1.1 getClassObject()

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

Implements Tower.

6.4.1.2 shoot()

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

Implements Tower.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bulletTower.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/bulletTower.cpp

6.5 Button Class Reference

Inheritance diagram for Button:

class_button-eps-converted-to.pdf

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:

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/button.hpp

6.6 Enemy Class Reference

Inheritance diagram for Enemy:

class_enemy-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)
- 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
- void **updateHealthText** (const sf::Font &font)
- 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
- int **slowed_** =0
- int **money_**
- sf::Vector2f **velocity_**
- std::queue< sf::Vector2f > **waypoints_**
- sf::Vector2f **currentWaypoint_**
- int **direction_**

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/enemy.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/enemy.cpp

6.7 Explosion Class Reference

Small class for drawing bomb explosions.

`#include <explosion.hpp>`

Inheritance diagram for Explosion:

Public Member Functions

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

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

  Updates the explosion.
- bool **isDone** ()

  Return done_ which tells if the explosion is done.

Private Attributes

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

6.7.1 Detailed Description

Small class for drawing bomb explosions.

See also

6.7.2 Constructor & Destructor Documentation

6.7.2.1 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

6.7.3 Member Function Documentation

6.7.3.1 update()

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

Updates the explosion.

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

Parameters

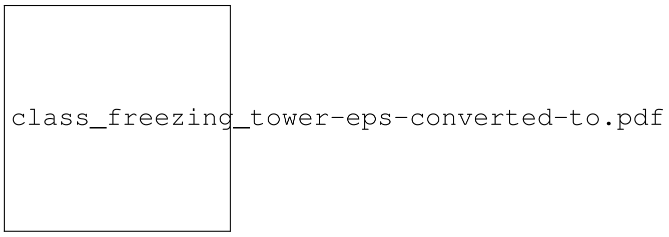
inputtime	Time between frames from Game::getTime()

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/explosion.hpp

6.8 FreezingTower Class Reference

Inheritance diagram for FreezingTower:



Public Member Functions

- **FreezingTower** (sf::Vector2f)
- void update (std::list< std::shared_ptr< Enemy > > &enemies) override
- Projectile ∗ shoot () override
- std::shared_ptr< Tower > getClassObject ()

Public Member Functions inherited from Tower

- **Tower** (sf::Vector2f position, const std::string &type="Basic", int baseCost=100, float range=100.0, float fire↩ Rate=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 maxLevel↩ Reached=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** ()
- void **upgradeTower** ()
- bool **isActive** ()
- void **unactiveHUD** ()
- void **activateHUD** ()
- sf::Vector2f **getSize** ()
- virtual void **build** ()

Private Attributes

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

Additional Inherited Members

Public Attributes inherited from Tower

- bool **HUDactive** = false
- bool **builded** = false

6.8.1 Member Function Documentation

6.8.1.1 getClassObject()

```
std::shared_ptr< Tower > FreezingTower::getClassObject ( )  [virtual]
```

Implements Tower.

6.8.1.2 shoot()

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

Implements Tower.

6.8.1.3 update()

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

Reimplemented from Tower.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/freezingTower.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/freezingTower.cpp

6.9 Game Class Reference

Public Member Functions

- Game ()
- void **run** ()
- path & **getPath** ()

Public Attributes

- Map **map**

Private Member Functions

- void **processEvents** ()
- void update ()
- void **render** ()
- void **addTower** (const Tower &tower)
- void **addEnemy** (const Enemy &enemy)
- void **addProjectile** (const Projectile &projectile)
- void **createPath** ()
- void **checkTowers** ()
- void **testEnemy** ()
- void **testEnemySplit** (sf::Vector2f position, std::queue< sf::Vector2f > waypoints)
- 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_**
- Menu ∗ **shop_**
- Menu ∗ **alternativeMenu_**
- Tower ∗ **activeTower_**
- 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 **Menu**
- class **LevelManager**

6.9.1 Constructor & Destructor Documentation

6.9.1.1 Game()

```
Game::Game ( )
```

!! get castle to be at end of path, getWaypoints.back() puts it in a stragne position on all paths

6.9.2 Member Function Documentation

6.9.2.1 update()

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

depending wether game or palyer keeps track of castle position atl: castle_.getGlobalBounds().intersects(enemy.↩ getGlobalBounds())){ player_.removeHP(10) <– should prob be enemy specific

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/game.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/game.cpp

6.10 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 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.
  Map keys:

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

6.10.1 Constructor & Destructor Documentation

6.10.1.1 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 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()

6.10.2 Member Function Documentation

6.10.2.1 readingSuccessfull()

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

returns status flag for reading level info from file.
True if reading was successfull, false if not

6.10.2.2 readLevels()

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

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

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

6.10.2.3 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 are waves left for the level. Moves to a new level once previous is complete and there are no enemies left.

See also

initiateEnemies()

6.10.3 Member Data Documentation

6.10.3.1 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.
Map 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:

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/levelManager.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/levelManager.cpp

6.11 Map Class Reference

Inheritance diagram for Map:

class_map-eps-converted-to.pdf

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**

Private Member Functions

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

Private Attributes

- std::vector< std::shared_ptr< Tower > > **towers**

Friends

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

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/map.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/map.cpp

6.12 Menu Class Reference

Class for storing a collection of buttons, a menu.

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

Public Member Functions

- void draw (sf::RenderWindow &window)

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

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

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

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

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

  Draws active tower range.

Private Member Functions

- void newTower (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::list< sf::Text > **texts_**
- sf::RectangleShape **bg_**

6.12.1 Detailed Description

Class for storing a collection of buttons, a menu.

6.12.2 Member Function Documentation

6.12.2.1 canBePlaced()

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

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

Parameters

game	Pointer to the game object

Returns

true, if the tower can be placed

6.12.2.2 checkButtons()

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

Checks if a button in the menu has been pressed.

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

Parameters

game	Pointer to the game object

6.12.2.3 createMenu()

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

Creates the buttons and texts of a menu.

Parameters

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

6.12.2.4 drag()

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

Implements drag&drop placing of towers.

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

Parameters

game	pointer to the game object

See also

canBePlaced()

6.12.2.5 draw()

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

Draws all the objects in the menu.

Parameters

window	window onto which the objects get drawn

6.12.2.6 drawRange()

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

Draws active tower range.

Parameters

game	pointer to the game object

6.12.2.7 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

6.12.2.8 update()

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

Updates the status of the menu.

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

Parameters

player	Reference to the player object

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

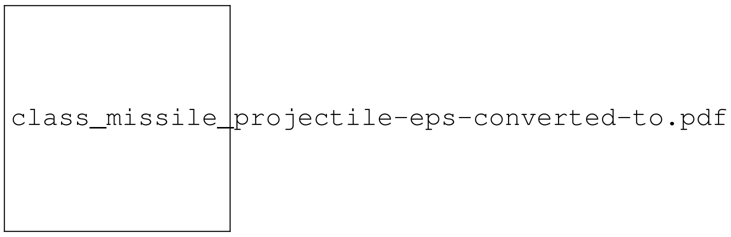
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/menu.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/menu.cpp

6.13 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 ()

Public Member Functions inherited from Projectile

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

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

    Destroy the Projectile object.
- 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 needs 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.

Private Attributes

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

6.13.1 Detailed Description

a projectile that targets (follows) a specific enemy

6.13.2 Constructor & Destructor Documentation

6.13.2.1 MissileProjectile()

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

Missile 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

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

6.13.3 Member Function Documentation

6.13.3.1 hasHitEnemy()

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

Parameters

enemy	is a reference to an Enemy object

Implements Projectile.

6.13.3.2 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.

6.13.3.3 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 is 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.

Parameters

game	is a reference to the running game instance

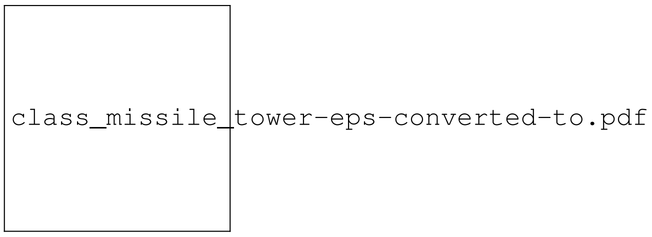
Implements Projectile.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/missileProjectile.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/missileProjectile.cpp

6.14 MissileTower Class Reference

Inheritance diagram for MissileTower:



Public Member Functions

- **MissileTower** (sf::Vector2f)
- MissileProjectile ∗ shoot () override
- std::shared_ptr< Tower > getClassObject () override

Public Member Functions inherited from Tower

- **Tower** (sf::Vector2f position, const std::string &type="Basic", int baseCost=100, float range=100.0, float fire↩ Rate=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 maxLevel↩ Reached=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** ()
- void **upgradeTower** ()
- virtual void **update** (std::list< std::shared_ptr< Enemy > > &enemies)
- bool **isActive** ()
- void **unactiveHUD** ()
- void **activateHUD** ()
- sf::Vector2f **getSize** ()
- virtual void **build** ()

Additional Inherited Members

Public Attributes inherited from Tower

- bool **HUDactive** = false
- bool **builded** = false

6.14.1 Member Function Documentation

6.14.1.1 getClassObject()

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

Implements Tower.

6.14.1.2 shoot()

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

Implements Tower.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/missileTower.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/missileTower.cpp

6.15 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**

6.15.1 Constructor & Destructor Documentation

6.15.1.1 path()

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

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

Parameters

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

See also

readPath()

6.15.2 Member Function Documentation

6.15.2.1 readingSuccessfull()

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

Returns

6.15.2.2 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:

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/path.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/path.cpp

6.16 Player Class Reference

Inheritance diagram for Player:

Public Member Functions

- int **getWallet** () const

    returns how much money the player has
- int **getHP** () const

    returns how many health points the player has
- 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

Private Attributes

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

6.16.1 Member Function Documentation

6.16.1.1 addMoney()

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

adds money to the players wallet

Parameters

amount	is how much money is to be added

6.16.1.2 removeHP()

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

removes health pointe from the player

Parameters

amount	is how much hp is to be removed

6.16.1.3 removeMoney()

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

removes money from the players wallet

Parameters

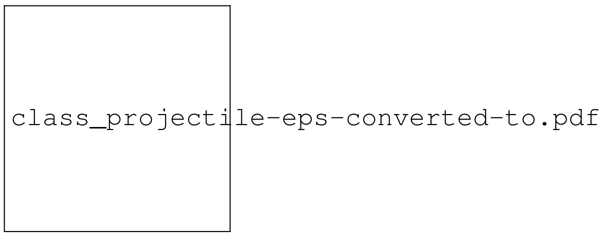
cost	is how much money is to be removed

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/player.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/player.cpp

6.17 Projectile Class Reference

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 position.
- virtual ∼**Projectile** ()

  Destroy the Projectile object.
- 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 needs 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 > &)=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_**

6.17.1 Constructor & Destructor Documentation

6.17.1.1 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 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, pre-definened for each derived type

6.17.2 Member Function Documentation

6.17.2.1 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.

6.17.2.2 hasHitEnemy()

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

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

Implemented in BulletProjectile, MissileProjectile, and BombProjectile.

6.17.2.3 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.

6.17.2.4 update()

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

updates the projectiles state as is defiened in each derived class

Implemented in BombProjectile, BulletProjectile, and MissileProjectile.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/projectile.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/projectile.cpp

6.18 ResourceContainer< T_enum, T_resource > Class Template Reference

Template container for textures etc resources.

`#include <resource_container.hpp>`

Public Member Functions

- void load (T_enum type, std::string filename)

  Loads and stores a resource.
- T_resource & get (T_enum type) const

  Find and return requested resource.

Private Attributes

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

6.18.1 Detailed Description

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

Template container for textures etc resources.

6.18.2 Member Function Documentation

6.18.2.1 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

6.18.2.2 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

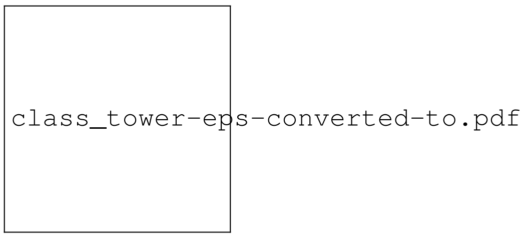
type	Enumerator which defines the type of this resource.
filename	path to file containing the resource.

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/resource_container.hpp

6.19 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 **builded** = false

Private Attributes

- const std::string **type_**
- const int **baseCost_**
- const float **range_**
- int **damage_**
- const float **fireRate_**
- int **currentLvl_**
- const CanDamage **damageType_**
- const int **upgradeCost_**
- std::shared_ptr< Enemy > **lockedEnemy_**
- sf::Clock **fireTimer_**
- bool **maxLevelReached_**

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

- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/tower.hpp
- /Users/ellenmolin/Desktop/tower-defense-tran-duong-2/src/tower.cpp

Chapter 7

File Documentation

7.1 bombProjectile.hpp

```
#ifndef BOMB_PROJECTILE
#define BOMB_PROJECTILE

#include "projectile.hpp"
#include <list>

class BombProjectile : public Projectile
{
private:
    int blastRange_;
public:

    BombProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range) // <- tbd
    : Projectile(shootDirection, position, damage, 60.0, "bomb", range), blastRange_(1000) {}
                                        //  ^            ^  ^  tbd

    bool hasHitEnemy(std::shared_ptr<Enemy>& enemy) override;

    void update(Game& game);

    Textures::ProjectileID textureType(){ return Textures::Bomb; }
};


#endif
```

7.2 bombTower.hpp

```
#ifndef BOMB_TOWER_H
#define BOMB_TOWER_H
#include "tower.hpp"
#include "bombProjectile.hpp"
class BombTower : public Tower {
public:
    BombTower(sf::Vector2f);
    void update(std::list<std::shared_ptr<Enemy» &enemies) override;
    BombProjectile* shoot() override;
    std::shared_ptr<Tower> getClassObject() override;
private:
//    void draw() override;
// Draw method is inherited from sf::Sprite so I have commented this out for now
// If we want to include some animation stuff I’m not sure where it would be best to do
};
#endif //BOMB_TOWER_H
```

7.3 bulletProjectile.hpp

```
#ifndef BULLET_PROJECTILE
```

```
#define BULLET_PROJECTILE

#include "projectile.hpp"

class BulletProjectile : public Projectile
{
public:
    BulletProjectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float range)
    : Projectile(shootDirection, position, damage, 500, "bullet", range) {}

    bool hasHitEnemy(std::shared_ptr<Enemy>& enemy);

    void update(Game& game);

    Textures::ProjectileID textureType(){ return Textures::Bullet; }

    float rotationAngle() const; //this one is used to calculate rotation angle of a projectile.
};


#endif
```

7.4 bulletTower.hpp

```
#ifndef BULLET_TOWER_H
#define BULLET_TOWER_H
#include "tower.hpp"
#include "bulletProjectile.hpp"
class BulletTower : public Tower {
public:
    BulletTower(sf::Vector2f);
    /*update() method for BulletTower uses update() from base class,
      hence I have commented it out*/
    //void update(std::list<std::shared_ptr<Enemy» &enemies) override;
    BulletProjectile* shoot() override;
    std::shared_ptr<Tower> getClassObject() override;
private:
    //void draw() override;
};
#endif //BULLET_TOWER_H
```

7.5 button.hpp

```
#ifndef BUTTON
#define BUTTON
#include <SFML/Graphics.hpp>

enum class Actions{
    Tower1,
    Tower2,
    Tower3,
    Tower4,
    Tower5,
    Pause,
    Upgrade,
    Sell,
    Close,  // In upgrade menu, closes upgrade menu.
    Level   // Click to start level
};

class Button : public sf::Sprite {
public:
    Button(Actions action, sf::Texture& texture, sf::Vector2f position, std::string text, sf::Font&
font) : action_(action) {
        setTexture(texture);
        setPosition(position);
        label_ = sf::Text(text, font, 15);
        label_.setPosition(position.x, position.y+20);
    }
    bool isClicked(sf::Vector2f mousePos) const {
        return getGlobalBounds().contains(mousePos);
    }

    Actions getAction() const {return action_;}
    sf::Text getLabel() const {return label_;}

private:
    Actions action_;
    sf::Text label_;
```

```cpp

};


#endif
```

7.6 enemy.hpp

```cpp
#ifndef ENEMY_HPP
#define ENEMY_HPP
#include <string>
#include "path.hpp"
#include <queue>
#include "player.hpp"
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics.hpp>
#include <random>

enum class EnemyType {
    Ground,
    Flying,
    Split,
};

class Enemy :public sf::Sprite {
public:

    Enemy(int hp, int speed, EnemyType type, int money, std::queue<sf::Vector2f> waypoints)
         : hp_(hp), actualSpeed_(speed), speed_(speed), effectiveSpeed_(speed), type_(type),
money_(money), waypoints_(waypoints), initialHp_(hp) {

        // Random y value of starting pos, gets set as a negative value
        // So enemies spawn outside window and then move in
        //int rand_y = std::rand() % 40;

        //tries to avoid enemies being on top of eachother
        std::random_device rd;
        std::uniform_int_distribution range(1,40);
        int x = range(rd);
        int y = range(rd);
        setPosition(waypoints_.front() - sf::Vector2f(x,y));

        if (!waypoints_.empty()) {
            currentWaypoint_ = waypoints_.front();
        }
        setVelocity();
    }



    ~Enemy() {}

    void update(sf::Time time); //update the state of the monster in relation to the game

    sf::Vector2f getCenter();

    sf::Vector2f getLocation();

    bool dead();

    int hp();

    int initialHp();

    float speed();

    int poisonStatus();

    int slowedStatus();

    EnemyType type();

    void takeDamage(int damage); //decreases the hp_ variable and if hp reaches 0 than the enemy is
automatically destroyed

    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;

    void updateHealthText(const sf::Font& font);

    const sf::Text& getHealthText() const;

private:
    int hp_;

    int initialHp_;

    bool dead_= false;

    float speed_;

    float actualSpeed_;

    float effectiveSpeed_;

    sf::Text healthText_;

    EnemyType type_;
         //has reference to player instance so money can be deposited to the player as well as the use
of other player functions

    int poison_=0; //If poison is larger than 0 that means that the enemy is poisoned
    // the length of time that the enemy is poisoned for depends on how large the poison
    //value is as the number decreases incrimently until 0
    int slowed_=0;
    //How much money the player recieves for killing the monster
    int money_;
    //waypoint based movement, the path class provides a queue of waypoints that take the enemies
through the path to the end

    sf::Vector2f velocity_;

    std::queue<sf::Vector2f> waypoints_;

    sf::Vector2f currentWaypoint_;

    int direction_; //0 = down, 1= left, 2= right, 3 = up

};

#endif
```

7.7 explosion.hpp

```
#ifndef EXPLOSION
#define EXPLOSION
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <stdio.h>
#define BOMB_SIZE_HALF 24

class Explosion : public sf::CircleShape {
public:
    Explosion(int blastRange, sf::Vector2f pos) : blastRange_(blastRange), done_(false) {
        time_ = sf::seconds(1);
        setPosition(pos.x + BOMB_SIZE_HALF, pos.y + BOMB_SIZE_HALF);
        setRadius(2);
        setOrigin(2, 2);
        setFillColor(sf::Color(255, 64, 0, 150));
    }

    void update(sf::Time inputtime) {
        time_ -= inputtime;
        if (time_ < sf::microseconds(0)) {
            done_ = true;
            std::cout « "The explosion is done" « std::endl;
            return;
        }
        if (time_ >= sf::seconds(0.5)) {
            setScale(getScale().x + 1, getScale().y + 1);
        } else {
            setScale(getScale().x - 1, getScale().y - 1);
        }

    }

    bool isDone(){ return done_; }


private:
    sf::Time time_;
    int blastRange_;
    bool done_;

};

#endif
```

7.8 freezingTower.hpp

```
#ifndef FREEZING_TOWER
#define FREEZING_TOWER
#include "tower.hpp"
#include "enemy.hpp"
#include <list>
#include <memory>

class FreezingTower : public Tower{
public:
    FreezingTower(sf::Vector2f);
    void update(std::list<std::shared_ptr<Enemy» &enemies) override;
    Projectile* shoot() override;
    std::shared_ptr<Tower> getClassObject();
private:
    std::list<std::shared_ptr<Enemy» lockedEnemies_;
};
#endif //FREEZING_TOWER
```

7.9 game.hpp

```
#ifndef GAME_HPP
#define GAME_HPP

#include <SFML/Graphics.hpp>
#include <list>
#include "tower.hpp"
#include "path.hpp"
#include "enemy.hpp"
#include "projectile.hpp"
#include "resource_container.hpp"
#include "player.hpp"
#include <memory> //for shared_ptr
#include "bulletTower.hpp"
#include "button.hpp"
#include "map.hpp"
#include "missileProjectile.hpp"
#include "menu.hpp"
#include <vector>
#include "levelManager.hpp"
#include "explosion.hpp"

class Menu;
// Class for running the game logic

class Game {

    friend class Tower;
    friend class BulletTower;
    friend class BombTower;
    friend class MissileTower;
    friend class FreezingTower;
    friend class BombProjectile;
    friend class BulletProjectile;
    friend class MissileProjectile;
    friend class Menu;
    friend class LevelManager;
```

```

public:
    Map map;
    Game();
    void run();

    ~Game(){

        for(auto i : enemies_){
            //delete i;
        }
        enemies_.clear();

        for(auto i : projectiles_){
            delete i;
        }
        projectiles_.clear();

        for(auto i : towers_){
            delete i;
        }
        towers_.clear();

        delete activeTower_;
        delete alternativeMenu_;
        delete shop_;
    }
    path& getPath();
private:
    void processEvents();
    void update();
    void render();
    void addTower(const Tower& tower);
    void addEnemy(const Enemy& enemy);
    void addProjectile(const Projectile& projectile);
    void createPath(); //this will create the path that the enemies will traverse (this should also be
rendered visually in the game)
    void checkTowers();
    void testEnemy();
    void testEnemySplit(sf::Vector2f position, std::queue<sf::Vector2f> waypoints);
    //adding a function to return the elapsed time
    sf::Time getTime() const;
    //I am adding a clock and time functionality that will need to be used for enemy movement and
updating and other game logic
    sf::Clock clock_;
    sf::Time time_;
    sf::RenderWindow window_;
    /* Pavel: should we change enemies_ to be
    * std::list<std::shared_ptr<Enemy» enemies_ instead and
    * initialize it with enemies using
    * enemies_.push_back(std::make_shared<TYPE_OF_ENEMY>(args))?
    * This way enemy would get destroyed automatically when
    * it is no longer locked by any tower AND it has reached 0 hp
    * (it gets removed from the list at this point) AND no projectile
    * flies towards it (we need to add shared_ptr<Enemy> member to projectile class).
    * There might be some error in my logic though....
    */
    std::list<Tower*> towers_;
    //std::list<Enemy> enemies_;

    /* Changed these to unque ptr, as looping over the abstract types directly
        is not possible, at least according to my understanding
        */
    std::list<std::shared_ptr<Enemy» enemies_;
    std::list<Projectile*> projectiles_;
    std::list<Explosion*> explosions_;
    path path_;
    std::list<Button> buttons_; // Stores clickable buttons
    bool dragged_; // Indicates if a tower is currently being dragged into place
    bool paused_; // Is the game paused?
    bool isGameOver_=false; //is the game over because the player has died to an enemy
    bool isGameFinished_ = false; //completed game
    sf::Font font_; // Stores text font
    sf::Text gameOverText;
    sf::Text gameFinishedText;
    sf::Sprite castle_sprite_;

    Menu* shop_; // Shop on left side
    Menu* alternativeMenu_; // stores menu for upgrading, beginning game, and advancing to next level
    Tower* activeTower_; // Pointer to tower that is being upgraded

    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_;
};

#endif
```

7.10 levelManager.hpp

```
#ifndef LEVELMANAGER
#define LEVELMANAGER

#pragma once

#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <variant>
#include <fstream>
#include <sstream>
#include <random>

#include "enemy.hpp"
#include "path.hpp"

class Game;

class LevelManager {

    public:
    //this requires c++17
    using variantData = std::variant<int, float, std::vector<int»;

    LevelManager(const std::string& src, path& path, Game& game, Player& player) : src_(src),
path_(path), game_(game), player_(player) {
        readLevels();

        currLevel_ = 0;
        waitTime_ = 0;
        levelTotal_ = levelSpecs_.size();
    }
    ~LevelManager(){}

    int getCurrentLevel();

    int getLevelTotal();

    void update();

    bool readingSuccessfull(); //need to abort game if reading fails, use this to check

    //void printLevelSpecs();

    private:

    void readLevels();

    void initiateEnemies();


    std::vector<std::map<std::string, variantData» levelSpecs_;

    int currLevel_;
    const std::string& src_;
    bool readingSuccess_;
    int levelTotal_;
    float waitTime_;

    path& path_;
    Game& game_;
    Player& player_;
};

#endif
```

7.11 map.hpp

```
#ifndef MAP_HPP
#define MAP_HPP
```

```

#include <SFML/Graphics.hpp>
#include <memory>
#include <string>
#include <vector>
#include "tower.hpp"

class Tower; // Forward declaration

class Map : public sf::Drawable, public sf::Transformable {
    friend class MainGame;
    friend class TowerManagement;



public:
    sf::Texture texture;
    sf::Sprite background;
    Map();
    ~Map();

    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);
    std::vector<sf::FloatRect> unBuildable;

    sf::FloatRect getBackgroundBounds();

private:
    std::vector<std::shared_ptr<Tower» towers;


    void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
};

#endif // MAP_HPP
```

7.12 menu.hpp

```
#ifndef MENU
#define MENU
#include <SFML/Graphics.hpp>
#include <list>
#include "button.hpp"
#include "game.hpp"
#include "tower.hpp"

// These are used in createMenu()
// the enum determines what type of menu is created:
// Which buttons are added etc.
enum class MenuType{
    Shop,
    Upgrade,
    Begin,
    Level
};
class Menu {
public:
    void draw(sf::RenderWindow& window);

    void checkButtons(Game* game);

    void createMenu(MenuType menu, Game* game);

    void update(Player& player);

    void drag(Game* game);

    void drawRange(Game* game);
private:

    void newTower(Tower* tower, Game* game);

    bool canBePlaced(Game* game);

    std::list<Button> buttons_;
    std::list<sf::Text> texts_;
    sf::RectangleShape bg_;
};

#endif
```

7.13 missileProjectile.hpp

```
#ifndef MISSILE_PROJECTILE
#define MISSILE_PROJECTILE

#include "projectile.hpp"

class MissileProjectile : public Projectile
{
private:
    std::shared_ptr<Enemy> targetEnemy_;

public:
    MissileProjectile(sf::Vector2f position, int damage, std::shared_ptr<Enemy> targetEnemy)
    : Projectile(sf::Vector2f(0,0), position, damage, 100.0, "missile", 70), targetEnemy_(targetEnemy)
{}

    bool hasHitEnemy(std::shared_ptr<Enemy>& enemy);

    void update(Game& game);
    Textures::ProjectileID textureType(){ return Textures::Missile; }
};


#endif
```

7.14 missileTower.hpp

```
#ifndef MISSILE_TOWER
#define MISSILE_TOWER
#include "tower.hpp"
#include "missileProjectile.hpp"
class MissileTower : public Tower {
public:
    MissileTower(sf::Vector2f);
    MissileProjectile* shoot() override;
    std::shared_ptr<Tower> getClassObject() override;
};
#endif
```

7.15 path.hpp

```
#ifndef PATH_HPP
#define PATH_HPP
#include <queue>
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics.hpp>
#include <vector>
#include <random>

class path {
    friend class enemy;
public:
    path(const std::string& src) : src_(src) {
        readPath();

        std::random_device rd;
        std::uniform_int_distribution<int> range(0, paths_.size()-1);

        auto gamePath = paths_[range(rd)];

        for(const auto& point: gamePath){
            addWaypoint(point);
        }

    }

    //creates a path and populates the waypoints queue with
    //all the waypoints required for the enemy class to traverse the path


    ~path() {

    }

    void readPath();

    bool readingSuccessfull();


    void addWaypoint(const sf::Vector2f& point);

    std::queue<sf::Vector2f> getWaypoints() const;
    void makeUnBuildablePath();
    static const float width;
    std::queue<sf::Vector2f> waypoints_;
    std::vector <sf::Vector2f> wayPoints;
    std::vector <sf::FloatRect> unBuildable;

    std::vector<std::vector<sf::Vector2f» paths_;

private:
    const std::string& src_;
    bool readingSuccess_;

};

#endif
```

7.16 player.hpp

```
#ifndef PLAYER
#define PLAYER

#include <string>
#include <list>
#include "enemy.hpp"
#include "tower.hpp"
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <memory>
#include "resource_container.hpp"

class Tower;
class Enemy;

class Player : public sf::Sprite
{
    private:
        int hp_;
        int wallet_;
        int level_;

    public:
        Player() : hp_(500), wallet_(1000), level_(0){}

        ~Player() {}

        int getWallet() const;

        int getHP() const;

        int getLevel() const;

        void levelUp();

        void addMoney(int amount);

        void removeMoney(int cost);

        void removeHP(int amount);
};

#endif
```

7.17 projectile.hpp

```
#ifndef PROJECTILE
#define PROJECTILE

#include "tower.hpp"
#include "player.hpp"
#include "enemy.hpp"
#include "resource_container.hpp"
#include <SFML/System/Vector2.hpp>
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/Graphics.hpp>
#include <memory>
```

```
#include <iostream>

class Game;
class Enemy;

// Removed inheritance of sf::Transformable.
// I checked that sf::Sprite inherits both Drawable and Transformable -Otto
class Projectile : public sf::Sprite
{
    private:
        float speed_;
        std::string type_;
        int damage_;
        sf::Vector2f position_; // of tower that created
        float maxDistance_;
        sf::Vector2f shootDirection_;
        bool isDestroyed_;

    public:

        Projectile(sf::Vector2f shootDirection, sf::Vector2f position, int damage, float speed,
std::string type, float maxDistance)
        : shootDirection_(shootDirection), position_(position), damage_(damage), speed_(speed),
type_(type), maxDistance_(maxDistance),
        isDestroyed_(false){
            this->setPosition(position_);
        }

        // shootdirection, position, damage comes from tower
        // speed, type, maxDistance come from derived classes
        // shootDirection needs to be normalised vector

        virtual ~Projectile() {}

        float getSpeed() const;

        const std::string& getType() const;

        int getDamage() const;

        sf::Vector2f getShootDir() const;
        //sf::Vector2f getVelocity() const;

        void destroy();

        bool isDestroyed();

        bool distToTower();

        virtual bool hasHitEnemy(std::shared_ptr<Enemy>&) = 0;

       virtual void update(Game&) = 0;

       virtual Textures::ProjectileID textureType() = 0;
};
#endif

```

7.18 resource_container.hpp

```
#ifndef RESOURCE_CONTAINER
#define RESOURCE_CONTAINER
#include <SFML/Graphics.hpp>
#include <string>
#include <memory>

// Enums for different textures
namespace Textures{

    // NOTE: these could also be stored in one big enum...
    enum TowerID {BulletTower, BombTower, MissileTower, FreezingTower, Tower5};
    enum EnemyID {Enemy1, Enemy2, Enemy3, Enemy4, Enemy5};
    enum ProjectileID{Bullet, Bomb, Missile};
    enum Various {Pause, Castle, Dirt};
}

template <typename T_enum, typename T_resource>
class ResourceContainer {
public:

    void load(T_enum type, std::string filename){
        std::unique_ptr<T_resource> resource(new T_resource());

        if (!resource->loadFromFile(filename)){
            //TODO: Handle texture loading error
        }
        // The function move should avoid creating a copy of the object recource, when inserting it
into the map
        resources_.insert(std::make_pair(type, std::move(resource)));
        }

    T_resource& get(T_enum type) const {
        auto wanted = resources_.find(type);
        return *wanted->second;
    }


private:
    std::map<T_enum, std::unique_ptr<T_resource>> resources_;


};

#endif
```

7.19 tower.hpp

```
#ifndef TOWER_H
#define TOWER_H
// EDIT: I think we should reduce max level of tower to lvl 2.
#define TOWER_MAX_LVL 2
#include <string>
#include <array>
#include <SFML/System/Vector2.hpp>
#include <SFML/System/Clock.hpp>
#include <SFML/Graphics.hpp>
#include "projectile.hpp"
#include "enemy.hpp"
#include <memory>

// enum class CanDamage is needed for implementing enemy-locking logic
// (i.e., which EnemyType can be locked and damaged by a specific type of tower).
class Projectile;

enum class CanDamage {
    Ground,
    Flying,
    Both
};
/* Base tower class will be abstract (i.e., no objects of base tower class are to be constructable)*/
class Tower : public sf::Sprite {
public:
    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);
    /*Tower(sf::Vector2f position);*/
    // I think there is really no need for copy constructor or copy assignment operator
    const std::string& getType() const {return type_;}
    //const sf::Vector2f getPosition() const {return position_;}
    const int getBaseCost() const {return baseCost_;}
    const float getFireRate() const {return fireRate_;}
    const CanDamage getDamageType() const {return damageType_;}
    const float getRange() const {return range_;}
    int getDamage() const {return damage_;}
//    const CanDamage getDamageType() const {return damageType_;}
    std::shared_ptr<Enemy> getLockedEnemy() const {return lockedEnemy_;}
    void setLockedEnemy(std::shared_ptr<Enemy> enemy) {lockedEnemy_ = enemy;}
    bool isMaxLevelReached() const {return maxLevelReached_;};
    int getCurrentLvl() const {return currentLvl_;}
    const int getUpgradeCost() const {return upgradeCost_;};
    sf::Clock getFireTimer() {return fireTimer_;}
    bool enemyWithinRange(std::shared_ptr<Enemy> enemy);
    void resetFireTimer() {fireTimer_.restart();}
    // shoot() creates a projectile that flies towards lockedEnemy_
    // Changed it to pure virtual
    virtual Projectile* shoot() = 0;
    void upgradeTower(); // Will be defined in .cpp
    /* update() method is declared as virtual. Some derived
       classes will use base update() and other will use override*/
    virtual void update(std::list<std::shared_ptr<Enemy>> &enemies);
    //This is what I add to support for the map class
    virtual std::shared_ptr<Tower> getClassObject() = 0; //Type of Tower
    bool isActive();//Whether the Tower is active or not
    bool HUDactive = false;//Temporary variable hold the state of the Tower
    void unactiveHUD();//Function to deactivate Tower (when sell)
    void activateHUD();//Function to activate Tower (when buy)
    sf::Vector2f getSize();//Get the height and width of the Tower we want to build
    virtual void build(); //build function which change the temporary variable builded to tru
    bool builded = false; //Temporary variable define whether the tower is built or not.
private:
//    virtual void draw();
    const std::string type_;
    //const sf::Vector2f position_;
    const int baseCost_;
    const float range_;
    int damage_;
    const float fireRate_; // Rate at which tower creates new projectiles; perhaps fireRate shouldn't
be upgradable, instead stronger projectiles are created
    int currentLvl_;
    const CanDamage damageType_;
    const int upgradeCost_;
    std::shared_ptr<Enemy> lockedEnemy_;
    sf::Clock fireTimer_;
    bool maxLevelReached_;
};
#endif //TOWER_H
```