Tower Defence

Generated by Doxygen 1.9.8

Chapter 1

Tower Defence

ELEC-7151 Object oriented programming software project fall 2023

Akseli Tuominen, Nandu Jagdish, Niilo Siren, Noel Nironen

Table of Contents

Overview

1.1 What the software does

In our program we successfully implemented all the basic features mentioned in the project topic description:

- A functioning tower defense game with basic graphics:
- Enemies follow a single, non-branched path
- Towers can shoot enemies inside their range
- Game is lost if any enemy reaches the end of the path
- Some money system, which gives more money per enemy killed and money is required to build towers
- Two modes: placing towers, running a wave of enemies through the path (towers cannot be moved when enemies are on the map)
- At least three different types of towers
  - A basic tower that shoots enemies within its range
  - A tower that slows down enemies within its range
  - A tower that has extra-long range
- At least three different types of enemies
  - A basic enemy
  - A tank enemy that has high health points, moves slow, and is immune to the tower slowing it down
  - An enemy that moves extra quickly
- At least five different levels with increasing difficulty
- Controlling the game by mouse: user can build/remove towers either between waves of enemies or without restrictions.
- Simple user interface that shows information such as resources, number of waves/enemies etc.

In addition to these basic features, we added the following additional features:

- Multiple enemy paths
- Our path has an intersection where the enemies choose the path randomly
- Additional tower that increases money
- Additional enemy type that can attack towers
- The map is read from a txt-file

Instructions for building and using the software

3.1.1 External Library Requirements

Needed Libraries for Linux (most computers have them all). Mac and Windows also have them

freetype

x11

xrandr

udev

opengl

flac

ogg

vorbis

vorbisenc

vorbisfile

openal

pthread

Downloading the libraries:

sudo apt update

sudo apt install \

```
libxrandr-dev \
libxcursor-dev \
libudev-dev \
libopenal-dev \
libflac-dev \
libvorbis-dev \
libgl1-mesa-dev \
libegl1-mesa-dev \
libdrm-dev \
libgbm-dev \
libfreetype-dev
```

Also CMake needs to be installed

3.1.2 Running the software

1. Clone the repository

2. Move to the repository and input the next commands to terminal

- cd tower-defense-tran-duong-5/build/

- cmake ..
- make
- cd bin
- ./CMakeSFMLProject

3.2 Software Usage Guide

After starting the game, you will see the main menu screen that consists of:

The difficulty levels that can be chosen by clicking

The play button that starts the game with the chosen difficulty

The exit button that closes the program

After starting a game, you will enter the building state, where you can build and remove towers. Building towers costs money and you can see the amount of money you have in the top right corner of the screen.

Towers:

Basic tower (red), cost 30

Slowing tower (blue), cost 50

Long range tower (yellow), cost 120

Money tower (black), costs 300

Building the tower is done by left-clicking the right color button and then right clicking a tile in which you want to place the tower; towers can be placed only on green tiles. By left-clicking a placed tower, it can be removed, and you will receive 50% of the cost back.

Leaving the building state and starting the attacking phase happens when the black button is pressed. After every enemy in the wave is defeated, a building phase will automatically start again.

The game ends when one enemy gets to the end of the path, and you will be shown an ending screen. By clicking the screen, you will get back to the main menu and a new game can be started.

Testing

Module Testing Due to the development nature of the project no third-party testing tool was used. Instead, the team relied on a combination of targeted print statements and the c++ gdb debugger. It can be said that the testing method employed was manual testing.

Testing Methods The primary testing method was using the GDB debugger integrated into the VSCode C++ development stack. This follows by adding breakpoints into the selected lines of code and checking whether the breakpoints are being hit and variable in interest in being modified or the desired function is being called.

Testing Outcomes During the course of the projects any failed test ie breakpoints that were not being hit or functions that was not being called, the call stack was used to determine the cause of the issue and fixed manually.

Work log

5.1 Division of Work and Responsibilities

At the beginning of the project, during the planning we divided the work and responsibilities for each member, but that was not very strictly followed and at our weekly meetings we picked a feature or subject for every member to work on. During the weekly (and towards the end, more frequent) meetings we merged our branches together for a working game and in addition to that we also tried to develop our game together as much as possible.

5.2 Weekly Work Descriptions

Week 1

Akseli: Working on the project plan(4h)

Nandu: Working on the project plan(1h)

Niilo: Working on the project plan(1h)

Noel: Working on the project plan (2h)

Week 2

Akseli: Created first map and the tile system. (4h)

Nandu: Formulated the attacking mechanism(2h)

Niilo: Learning SFML graphics and built-in functions added enemy class (4h)

Noel: Debugging SFML and CMake problems (10h)

Week 3

Akseli: Towers and tower placement implemented. (6h)

Nandu: Implemented the tower attack mechanism (3h)

Niilo: Initial timestep system with sleep function (4h)

Noel: SFML problems, enemy class and enemy spawning (9h)

Week 4

Akseli: Game flow created with game states and Main menu implemented. (4h)

Nandu: Implemented clock functions (3h)

Niilo: Initial enemy movement and testing (6h)

Noel: Main menu and graphics (5h)

Week 5

Akseli: Losing/End screen implemented and debugging the game states. Also texts for buttons (2h)

Nandu: Fixed enemy destruction and tower range (3h)

Niilo: Enemy movement following one path (5h)

Noel: Added money system (6h)

Week 6

Akseli: Added different kinds of towers, one that slows, one normal and one with huge range. Also Added the option to remove towers.(7h)

Nandu: Added tower cost and enemy points (3h)

Niilo: Added multiple enemies and their classes (3h)

Noel: Improved the money system (2h)

Week 7

Akseli: Fixed towers so that they only attack the first enemy. Created helper add button and text functions.(3h)

Nandu: Fixed money system (2h)

Niilo: Level system implemented, game resetting, difficulty system (4h)

Noel:

Week 8

Akseli: Added Tutorial, maps now load from file, basic fixes in the code and game balancing, edited Cmake and figured out what is needed to run the program. (16h)

Nandu: Added documentation and cleanup of code.

Niilo: Added 4th Enemy type, added cash Tower, balanced game. Enemy formation updated, added tower prices, enemy movement trough two paths with some intelligence (18h)

Noel: Improving money system, multiple enemy paths, documentation, configurating the game building and compiling (16h)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 Enemy Class Reference

Enemy class.

`#include <enemies.h>`

Public Member Functions

- Enemy (sf::Vector2f &position, double radius, int health, double speed, float x, float y, sf::Color &color, int points)

    Constructs an enemy object with the given parameters.
- sf::CircleShape & getShape ()
- sf::Vector2f & getPosition ()
- void move (float x_dir, float y_dir)
- void moveEnemy (double timeStep, sf::RenderWindow &window)
- int getRoute ()
- int getSpeed ()
- int getXcoord ()
- int getYcoord ()
- void addY (int b)
- void addX (int a)
- void lowerHealth (int h)
- bool hasPassed ()
- bool isDead ()
- void getHit (int damage)
- int getHealth ()
- int getPoints ()
- void reduceSpeed ()
- ∼Enemy ()

5.1.1 Detailed Description

Enemy class.

Represents an enemy object in the game.

The Enemy class encapsulates the properties and behaviors of an enemy in the game. It includes information such as position, health, speed, and points awarded for defeating the enemy. Enemies can move, take damage, and have their speed reduced based on their fill color.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Enemy()

```
Enemy::Enemy (
             sf::Vector2f & position,
             double radius,
             int health,
             double speed,
             float x,
             float y,
             sf::Color & color,
             int points )
```

Constructs an enemy object with the given parameters.

Enemy.cpp.

Parameters

position	The initial position of the enemy.
radius	The radius of the enemy.
health	The initial health of the enemy.
speed	The speed of the enemy.
x	The x-coordinate of the enemy's position.
y	The y-coordinate of the enemy's position.
color	The color of the enemy.
points	The points awarded for defeating the enemy.

5.1.2.2 ∼Enemy()

```
Enemy::∼Enemy ( )  [inline]
```

5.1.3 Member Function Documentation

5.1.3.1 addX()

```
void Enemy::addX (
             int a )
```

5.1.3.2 addY()

```
void Enemy::addY (
             int b )
```

5.1.3.3 getHealth()

```
int Enemy::getHealth ( )  [inline]
```

5.1.3.4 getHit()

```
void Enemy::getHit (
             int damage )  [inline]
```

5.1.3.5 getPoints()

```
int Enemy::getPoints ( )  [inline]
```

5.1.3.6 getPosition()

```
sf::Vector2f & Enemy::getPosition ( )
```

5.1.3.7 getRoute()

```
int Enemy::getRoute ( )
```

5.1.3.8 getShape()

```
sf::CircleShape & Enemy::getShape ( )
```

5.1.3.9 getSpeed()

```
int Enemy::getSpeed ( )
```

5.1.3.10 getXcoord()

```
int Enemy::getXcoord ( )
```

5.1.3.11 getYcoord()

```
int Enemy::getYcoord ( )
```

5.1.3.12 hasPassed()

```
bool Enemy::hasPassed ( )
```

5.1.3.13 isDead()

```
bool Enemy::isDead ( )
```

5.1.3.14 lowerHealth()

```
void Enemy::lowerHealth (
             int h )
```

Decreases the health of the enemy by the specified amount.

Parameters

h	The amount to decrease the health by.

5.1.3.15 move()

```
void Enemy::move (
             float x_dir,
             float y_dir )
```

5.1.3.16 moveEnemy()

```
void Enemy::moveEnemy (
             double timeStep,
             sf::RenderWindow & window )
```

5.1.3.17 reduceSpeed()

```
void Enemy::reduceSpeed ( )  [inline]
```

Reduces the speed of the enemy based on its fill color. If the fill color is Cyan, the speed is set to 2. If the fill color is Black or Red, the speed is set to 1.

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

- src/Objects/enemies.h
- src/Objects/enemies.cpp

5.2 EnemyType Class Reference

The base class for different types of enemies in the game.

```
#include <EnemyType.h>
```

Inheritance diagram for EnemyType:

Public Member Functions

- virtual Enemy createEnemy (sf::Vector2f &position, float x, float y) const =0

    Creates an enemy of this type at the specified position.
- virtual ∼EnemyType ()=default

    Destructor for the EnemyType class.

5.2.1 Detailed Description

The base class for different types of enemies in the game.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ∼EnemyType()

```
virtual EnemyType::∼EnemyType ( )  [virtual], [default]
```

Destructor for the EnemyType class.

5.2.3 Member Function Documentation

5.2.3.1 createEnemy()

```
virtual Enemy EnemyType::createEnemy (
            sf::Vector2f & position,
            float x,
            float y ) const  [pure virtual]
```

Creates an enemy of this type at the specified position.

Parameters

position	The position of the enemy.
x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

The created enemy object.

Implemented in EnemyTypeA, EnemyTypeB, EnemyTypeC, and EnemyTypeD.

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

- src/Objects/EnemyType.h

5.3 EnemyTypeA Class Reference

```
#include <EnemyTypeA.h>
```

Inheritance diagram for EnemyTypeA:

Collaboration diagram for EnemyTypeA:

Public Member Functions

- Enemy createEnemy (sf::Vector2f &position, float x, float y) const override
  Creates an enemy of this type at the specified position.

Public Member Functions inherited from EnemyType

- virtual ~EnemyType ()=default

    Destructor for the EnemyType class.

5.3.1 Member Function Documentation

5.3.1.1 createEnemy()

```
Enemy EnemyTypeA::createEnemy (
             sf::Vector2f & position,
             float x,
             float y ) const  [override], [virtual]
```

Creates an enemy of this type at the specified position.

Parameters

position	The position of the enemy.
x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

The created enemy object.

Implements EnemyType.

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

- src/Objects/EnemyTypeA.h
- src/Objects/EnemyTypeA.cpp

5.4 EnemyTypeB Class Reference

EnemyTypeB.h.

```
#include <EnemyTypeB.h>
```

Inheritance diagram for EnemyTypeB:

Collaboration diagram for EnemyTypeB:

Public Member Functions

- Enemy createEnemy (sf::Vector2f &position, float x, float y) const override

    Creates an enemy of this type at the specified position.

Public Member Functions inherited from EnemyType

- virtual ∼EnemyType ()=default

  Destructor for the EnemyType class.

5.4.1 Detailed Description

EnemyTypeB.h.

5.4.2 Member Function Documentation

5.4.2.1 createEnemy()

```
Enemy EnemyTypeB::createEnemy (
            sf::Vector2f & position,
            float x,
            float y ) const  [override], [virtual]
```

Creates an enemy of this type at the specified position.

Parameters

position	The position of the enemy.
x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

The created enemy object.

Implements EnemyType.

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

- src/Objects/EnemyTypeB.h
- src/Objects/EnemyTypeB.cpp

5.5 EnemyTypeC Class Reference

EnemyTypeC.h.

```
#include <EnemyTypeC.h>
```

Inheritance diagram for EnemyTypeC:

Collaboration diagram for EnemyTypeC:

Public Member Functions

- Enemy createEnemy (sf::Vector2f &position, float x, float y) const override

    Creates an enemy of this type at the specified position.

Public Member Functions inherited from EnemyType

- virtual ~EnemyType ()=default

    Destructor for the EnemyType class.

5.5.1 Detailed Description

EnemyTypeC.h.

5.5.2 Member Function Documentation

5.5.2.1 createEnemy()

```
Enemy EnemyTypeC::createEnemy (
            sf::Vector2f & position,
            float x,
            float y ) const  [override], [virtual]
```

Creates an enemy of this type at the specified position.

Parameters

position	The position of the enemy.
x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

The created enemy object.

Implements EnemyType.

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

- src/Objects/EnemyTypeC.h
- src/Objects/EnemyTypeC.cpp

5.6 EnemyTypeD Class Reference

EnemyTypeD.h.

```
#include <EnemyTypeD.h>
```

Inheritance diagram for EnemyTypeD:

Collaboration diagram for EnemyTypeD:

Public Member Functions

- Enemy createEnemy (sf::Vector2f &position, float x, float y) const override

  Creates an enemy of this type at the specified position.

Public Member Functions inherited from EnemyType

- virtual ∼EnemyType ()=default

  Destructor for the EnemyType class.

5.6.1 Detailed Description

EnemyTypeD.h.

5.6.2 Member Function Documentation

5.6.2.1 createEnemy()

```
Enemy EnemyTypeD::createEnemy (
            sf::Vector2f & position,
            float x,
            float y ) const  [override], [virtual]
```

Creates an enemy of this type at the specified position.

Parameters

position	The position of the enemy.
x	The x-coordinate of the position.
y	The y-coordinate of the position.

Returns

The created enemy object.

Implements EnemyType.

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

- src/Objects/EnemyTypeD.h
- src/Objects/EnemyTypeD.cpp

5.7 Tile Class Reference

Represents a tile in the game.

Public Member Functions

- Tile (const sf::Vector2f &position, const sf::Color &color, int tileSize)

    Constructs a Tile object.
- sf::RectangleShape & getShape ()

    Retrieves the shape of the tile.
- sf::Vector2f & getPosition ()

    Retrieves the position of the tile.
- const sf::Color & getColor () const

    Retrieves the color of the tile.

5.7.1 Detailed Description

Represents a tile in the game.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Tile()

```
Tile::Tile (
             const sf::Vector2f & position,
             const sf::Color & color,
             int tileSize )  [inline]
```

Constructs a Tile object.

Parameters

position	The position of the tile.
color	The color of the tile.
tileSize	The size of the tile.

5.7.3 Member Function Documentation

5.7.3.1 getColor()

```
const sf::Color & Tile::getColor ( ) const  [inline]
```

Retrieves the color of the tile.

Returns

A constant reference to the color of the tile.

5.7.3.2 getPosition()

```
sf::Vector2f & Tile::getPosition ( )  [inline]
```

Retrieves the position of the tile.

Returns

A reference to the position of the tile.

5.7.3.3 getShape()

```
sf::RectangleShape & Tile::getShape ( )  [inline]
```

Retrieves the shape of the tile.

Returns

A reference to the shape of the tile.

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

- src/tiles.cpp

5.8 Tower Class Reference

```
#include <tower.h>
```

Public Member Functions

- Tower (const sf::Vector2f &position, const TowerType &type)
- sf::ConvexShape & getShape ()
- double getAttack_range () const
- int attackEnemy (std::vector< Enemy > &enemies)
- sf::CircleShape getAttackShape ()
- sf::Vector2f getPosition ()
- void addClock (UniversalClock &clock)
- TowerType getType () const
- ~Tower ()

5.8.1 Constructor & Destructor Documentation

5.8.1.1 Tower()

```
Tower::Tower (
            const sf::Vector2f & position,
            const TowerType & type )
```

5.8.1.2 ∼Tower()

```
Tower::∼Tower ( )  [inline]
```

5.8.2 Member Function Documentation

5.8.2.1 addClock()

```
void Tower::addClock (
             UniversalClock & clock )
```

5.8.2.2 attackEnemy()

```
int Tower::attackEnemy (
             std::vector< Enemy > & enemies )
```

5.8.2.3 getAttack_range()

```
double Tower::getAttack_range ( ) const
```

5.8.2.4 getAttackShape()

```
sf::CircleShape Tower::getAttackShape ( )
```

5.8.2.5 getPosition()

```
sf::Vector2f Tower::getPosition ( )
```

5.8.2.6 getShape()

```
sf::ConvexShape & Tower::getShape ( )
```

5.8.2.7 getType()

```
TowerType Tower::getType ( ) const  [inline]
```

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

- src/Objects/tower.h
- src/Objects/tower.cpp

5.9 TowerType Class Reference

Tower class.

```
#include <tower.h>
```

Public Member Functions

- TowerType (double radius, int damage, double attack_range, double attack_speed, const sf::Color &color, int cost)

  Constructs a TowerType object with the specified parameters.
- double getRadius () const

  Gets the radius of the tower.
- int getDamage () const

  Gets the damage inflicted by the tower.
- double getAttackRange () const

  Gets the attack range of the tower.
- double getAttackSpeed () const

  Gets the attack speed of the tower.
- const sf::Color & getColor () const

  Gets the color of the tower.
- int getCost () const

  Gets the cost of the tower.
- ∼TowerType ()

  Destructor for the TowerType object.

5.9.1 Detailed Description

Tower class.

Represents a type of tower in the tower defense game.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 TowerType()

```
TowerType::TowerType (
              double radius,
              int damage,
              double attack_range,
              double attack_speed,
              const sf::Color & color,
              int cost )
```

Constructs a TowerType object with the specified parameters.

Parameters

radius	The radius of the tower.
damage	The damage inflicted by the tower.
attack_range	The attack range of the tower.
attack_speed	The attack speed of the tower.
color	The color of the tower.
cost	The cost of the tower.

5.9.2.2 ∼TowerType()

```
TowerType::∼TowerType ( )  [inline]
```

Destructor for the TowerType object.

5.9.3 Member Function Documentation

5.9.3.1 getAttackRange()

```
double TowerType::getAttackRange ( ) const
```

Gets the attack range of the tower.

Returns

The attack range of the tower.

5.9.3.2 getAttackSpeed()

```
double TowerType::getAttackSpeed ( ) const
```

Gets the attack speed of the tower.

Returns

The attack speed of the tower.

5.9.3.3 getColor()

```
const sf::Color & TowerType::getColor ( ) const
```

Gets the color of the tower.

Returns

The color of the tower.

5.9.3.4 getCost()

```
int TowerType::getCost ( ) const  [inline]
```

Gets the cost of the tower.

Returns

The cost of the tower.

5.9.3.5 getDamage()

```
int TowerType::getDamage ( ) const
```

Gets the damage inflicted by the tower.

Returns

The damage inflicted by the tower.

5.9.3.6 getRadius()

```
double TowerType::getRadius ( ) const
```

Gets the radius of the tower.

Returns

The radius of the tower.

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

- src/Objects/tower.h
- src/Objects/tower.cpp

5.10 UniversalClock Class Reference

The UniversalClock class represents a clock used for timing in the game engine.

```
#include <gameEngine.hpp>
```

Public Member Functions

- UniversalClock ()

  Constructs a UniversalClock object.
- bool isDelayFinished (float delayTime)

  Checks if the specified delay time has passed.
- void restartClock ()

  Restarts the clock.

5.10.1 Detailed Description

The UniversalClock class represents a clock used for timing in the game engine.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 UniversalClock()

```
UniversalClock::UniversalClock ( )  [inline]
```

Constructs a UniversalClock object.

5.10.3 Member Function Documentation

5.10.3.1 isDelayFinished()

```
bool UniversalClock::isDelayFinished (
            float delayTime )  [inline]
```

Checks if the specified delay time has passed.

Parameters

delayTime	The delay time in milliseconds.

Returns

True if the delay time has passed, false otherwise.

5.10.3.2 restartClock()

```
void UniversalClock::restartClock ( )  [inline]
```

Restarts the clock.

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

- src/gameEngine.hpp

Chapter 6

File Documentation

6.1 README.md File Reference

6.2 src/gameEngine.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "Objects/enemies.h"
```

Include dependency graph for gameEngine.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class UniversalClock

  The UniversalClock class represents a clock used for timing in the game engine.

6.3 gameEngine.hpp

Go to the documentation of this file.

```
#ifndef GAME_ENGINE_HPP
#define GAME_ENGINE_HPP
#include <SFML/Graphics.hpp>
#include "Objects/enemies.h"



class UniversalClock {
public:
    UniversalClock() : clock() {}

    bool isDelayFinished(float delayTime) {
        return clock.getElapsedTime().asMilliseconds() >= delayTime;
    }

    void restartClock() {
        clock.restart();
    }

private:
    sf::Clock clock;
};
#endif
```

6.4 src/Graphics/graphicFunctions.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <iostream>
#include <fstream>
#include <random>
#include "../Objects/tower.h"
#include "../Objects/enemies.h"
#include "../tiles.cpp"
#include "../Objects/EnemyTypeA.h"
#include "../Objects/EnemyTypeB.h"
#include "../Objects/EnemyTypeC.h"
#include "../Objects/EnemyTypeD.h"
```

Include dependency graph for graphicFunctions.cpp: This graph shows which files directly or indirectly include this file:

Functions

- void drawTiles (sf::RenderWindow &window, const int tileSize, const int windowWidth, const int window↩ Height, int difficulty)

  Draws the tiles on the game window based on the provided map and tile size.
- sf::RectangleShape createButton (float x, float y, float width, float height, sf::Color color)
- sf::Text createText (float x, float y, std::string content, sf::Font &font, unsigned int size, sf::Color color)
- void addTower (sf::RenderWindow &window, Tile tile, TowerType type)
- void addEnemy (sf::RenderWindow &window, int tileSize, int x, int y, int gameLevel, int difficulty)

  Adds enemies to the game based on the game level and difficulty.
- void placeTower (sf::Event event, sf::RenderWindow &window, int &money)
- void drawTowers (sf::RenderWindow &window, int &money)
- void onlyDrawTowers (sf::RenderWindow &window)
- void mainMenu (sf::RenderWindow &window, int difficulty)
- void drawMoney (sf::RenderWindow &window, int money)

  Draws the current amount of money on the screen.
- void endScreen (sf::RenderWindow &window)
- void deleteTower (sf::Event event, sf::RenderWindow &window, int &money)

  Deletes a tower from the game.
- void tutorial (sf::RenderWindow &window)
- void drawWave (sf::RenderWindow &window, int &gameLevel)

Variables

- std::vector< Tile > tiles
- std::vector< Tower > towers
- std::vector< Enemy > enemies
- sf::RectangleShape playButton
- sf::RectangleShape exitButton
- bool towerPlacementMode = false
- TowerType ∗ selectedTowerType = nullptr
- bool toMain = false

6.4.1 Function Documentation

6.4.1.1 addEnemy()

```
void addEnemy (
            sf::RenderWindow & window,
            int tileSize,
            int x,
            int y,
            int gameLevel,
            int difficulty )
```

Adds enemies to the game based on the game level and difficulty.

Parameters

window	The SFML RenderWindow object.
tileSize	The size of each tile in pixels.
x	The x-coordinate of the enemy's starting position.
y	The y-coordinate of the enemy's starting position.
gameLevel	The current game level.
difficulty	The difficulty level of the game.

6.4.1.2 addTower()

```
void addTower (
            sf::RenderWindow & window,
            Tile tile,
            TowerType type )
```

Adds a tower to the game window.

Parameters

window	The game window to add the tower to.
tile	The tile on which the tower is placed.
type	The type of tower to add.

6.4.1.3 createButton()

```
sf::RectangleShape createButton (
            float x,
            float y,
            float width,
            float height,
            sf::Color color )
```

6.4.1.4 createText()

```
sf::Text createText (
             float x,
             float y,
             std::string content,
             sf::Font & font,
             unsigned int size,
             sf::Color color )
```

Creates an sf::Text object with the specified properties.

Parameters

x	The x-coordinate of the text's position.
y	The y-coordinate of the text's position.
content	The content of the text.
font	The font to be used for the text.
size	The size of the text.
color	The color of the text.

Returns

The created sf::Text object.

6.4.1.5 deleteTower()

```
void deleteTower (
             sf::Event event,
             sf::RenderWindow & window,
             int & money )
```

Deletes a tower from the game.

This function takes an SFML event, a reference to the game window, and a reference to the player's money. It removes the tower from the game and updates the player's money accordingly.

Parameters

event	The SFML event that triggered the tower deletion.
window	The game window.
money	The player's money.

6.4.1.6 drawMoney()

```
void drawMoney (
             sf::RenderWindow & window,
             int money )
```

Draws the current amount of money on the screen.

Parameters

window	The SFML render window to draw on.
money	The current amount of money.

6.4.1.7 drawTiles()

```
void drawTiles (
             sf::RenderWindow & window,
             const int tileSize,
             const int windowWidth,
             const int windowHeight,
             int difficulty )
```

Draws the tiles on the game window based on the provided map and tile size.

Parameters

window	The game window to draw the tiles on.
tileSize	The size of each tile in pixels.
windowWidth	The width of the game window in pixels.
windowHeight	The height of the game window in pixels.
difficulty	The difficulty level of the game.

6.4.1.8 drawTowers()

```
void drawTowers (
             sf::RenderWindow & window,
             int & money )
```

Draws the towers on the specified window.

Parameters

window	The SFML RenderWindow to draw the towers on.
money	A reference to the money variable.

6.4.1.9 drawWave()

```
void drawWave (
             sf::RenderWindow & window,
             int & gameLevel )
```

Draws the wave of enemies on the game window.

Parameters

window	The SFML RenderWindow object to draw on.
gameLevel	The current level of the game.

6.4.1.10 endScreen()

```
void endScreen (
            sf::RenderWindow & window )
```

6.4.1.11 mainMenu()

```
void mainMenu (
            sf::RenderWindow & window,
            int difficulty )
```

Displays the main menu of the game.

Parameters

window	The SFML RenderWindow object used for rendering.
difficulty	The difficulty level of the game.

6.4.1.12 onlyDrawTowers()

```
void onlyDrawTowers (
            sf::RenderWindow & window )
```

6.4.1.13 placeTower()

```
void placeTower (
            sf::Event event,
            sf::RenderWindow & window,
            int & money )
```

Places a tower on the screen based on the given event.

Parameters

event	The event that triggered the tower placement.
window	The SFML render window.
money	The current amount of money the player has.

6.4.1.14 tutorial()

```
void tutorial (
            sf::RenderWindow & window )
```

6.4.2 Variable Documentation

6.4.2.1 enemies

```
std::vector<Enemy> enemies
```

6.4.2.2 exitButton

```
sf::RectangleShape exitButton
```

6.4.2.3 playButton

```
sf::RectangleShape playButton
```

6.4.2.4 selectedTowerType

```
TowerType* selectedTowerType = nullptr
```

6.4.2.5 tiles

```
std::vector<Tile> tiles
```

6.4.2.6 toMain

```
bool toMain = false
```

6.4.2.7 towerPlacementMode

```
bool towerPlacementMode = false
```

6.4.2.8 towers

```
std::vector<Tower> towers
```

6.5 graphicFunctions.cpp

Go to the documentation of this file.

```
#include <SFML/Graphics.hpp>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <iostream>
#include <fstream>
#include <random>
#include "../Objects/tower.h"
#include "../Objects/enemies.h"
#include "../tiles.cpp"
#include "../Objects/EnemyTypeA.h"
#include "../Objects/EnemyTypeB.h"
#include "../Objects/EnemyTypeC.h"
#include "../Objects/EnemyTypeD.h"

std::vector<Tile> tiles;
std::vector<Tower> towers;
std::vector<Enemy> enemies;
sf::RectangleShape playButton;
sf::RectangleShape exitButton;
bool towerPlacementMode = false;
TowerType* selectedTowerType = nullptr;
bool toMain = false;
//function to draw all the tiles from hardcoded map
void drawTiles(sf::RenderWindow &window, const int tileSize, const int windowWidth, const int
windowHeight,int difficulty) {
    //Count how many tiles we can fit in map
    const int mapWidth = windowWidth / tileSize;
    const int mapHeight = windowHeight / tileSize;


    // Define a hardcoded map, 0 for water(blue), 1 for grass(green) and other for path(white)
    int map[16][12];
    std::ifstream mapFile;
    if(difficulty > 3){
        mapFile.open("src/assets/map2.txt");
    }
    else{
        mapFile.open("src/assets/map1.txt");
    }

    if (!mapFile) {
        std::cerr « "Unable to open map file";
        // handle error
    }

    for (int i = 0; i < 16; ++i) {
        for (int j = 0; j < 12; ++j) {
            char ch;
            if (!(mapFile » ch)) {
                //std::cerr « "Error reading map file";
                // handle error
            }
            map[i][j] = ch - '0';  // convert char to int
        }
    }

    mapFile.close();

    //iterate through all the tiles
    for (int x = 0; x < mapWidth; x++) {
        for (int y = 0; y < mapHeight; y++) {
            //calculate position
            sf::Vector2f tilePosition(x * tileSize, y * tileSize);

            // Create a Tile object with the specified color, and tileSize
            int tileType = map[x][y];
            sf::Color tileColor;
            if (tileType == 0) {
                tileColor = sf::Color::Blue;
            } else if (tileType == 1) {
                tileColor = sf::Color::Green;
            } else {
                tileColor = sf::Color::White;
            }
            //create tile objects for each tile
            Tile tile(tilePosition, tileColor, tileSize);

            tiles.push_back(tile);
            // Draw the tile's shape
            window.draw(tile.getShape());
        }
    }
}
sf::RectangleShape createButton(float x, float y, float width, float height, sf::Color color) {
    sf::RectangleShape button(sf::Vector2f(width, height));
    button.setPosition(x, y);
    button.setFillColor(color);
    return button;
}

// Function to create a text
sf::Text createText(float x, float y, std::string content, sf::Font& font, unsigned int size,sf::Color color) {
    sf::Text text;
    text.setFont(font);
    text.setFillColor(color);
    text.setString(content);
    text.setCharacterSize(size);
    text.setPosition(x, y);
    return text;
}

void addTower(sf::RenderWindow &window, Tile tile, TowerType type){
    Tower tower(tile.getPosition(),type);
        towers.push_back(tower);
}

void addEnemy(sf::RenderWindow &window, int tileSize, int x, int y, int gameLevel, int difficulty){

    int iterator = gameLevel * difficulty;

    if (gameLevel <= 2) {
        for (int j = 1; j < 3*iterator; j++) {

            sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
            EnemyTypeA enemyTypeA;
            enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
        }
    }
    else if (gameLevel > 2 && gameLevel <= 4) {
        for (int j = 1; j < 3*iterator; j++) {
            sf::Vector2f tileStartPosition_B((-j-5)*tileSize+7, y * tileSize+7);
            EnemyTypeB enemyTypeB;
            enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B, (-j-5)*tileSize,y*tileSize));
        }
        for (int j = 1; j < 3*iterator; j++) {
            sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
            EnemyTypeA enemyTypeA;
            enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
        }
    }
    else if (gameLevel > 4 && gameLevel <= 6) {
        sf::Vector2f tileStartPosition_D((x-10.5)*tileSize+1, y * tileSize+1);
        EnemyTypeD enemyTypeD;
        enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-10.5)*tileSize,y*tileSize));

        for (int j = 1; j < 4*iterator; j++) {
            sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
            EnemyTypeB enemyTypeB;
            enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B, (-j-15)*tileSize,y*tileSize));
        }
        for (int j = 1; j < 2*iterator; j++){
            sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
            EnemyTypeA enemyTypeA;
            enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j-6)*tileSize,y*tileSize));
        }
        for (int j = 1; j < 1*iterator; j++){
            sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
            EnemyTypeC enemyTypeC;
            enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C, (-j-2)*tileSize,y*tileSize));
        }
    }
    else if (gameLevel > 6) {
        sf::Vector2f tileStartPosition_D((x-11.5)*tileSize+1, y * tileSize+1);
        EnemyTypeD enemyTypeD;
        enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-11.5)*tileSize,y*tileSize));

        for (int j = 1; j < 8*iterator; j++) {
            sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
            EnemyTypeB enemyTypeB;
            enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B, (-j-15)*tileSize,y*tileSize));
        }
        for (int j = 1; j < 4*iterator; j++){
            sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
            EnemyTypeA enemyTypeA;
            enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j-6)*tileSize,y*tileSize));
        }
        for (int j = 1; j < 2*iterator; j++){
            sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
            EnemyTypeC enemyTypeC;
            enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C, (-j-2)*tileSize,y*tileSize));
        }
    }


}



void placeTower(sf::Event event, sf::RenderWindow &window, int &money){

    sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }

    //Define the tower types
    TowerType basicTower(30.0, 20, 100, 50.0, sf::Color::Red,30);
    TowerType advancedTower(40.0, 30, 100, 60.0, sf::Color::Blue,50);
    TowerType ultimateTower(50.0, 40, 250, 70.0, sf::Color::Yellow,120);
    TowerType cashTower(0,0,100,50,sf::Color::Black,300);

    // Create the buttons for each tower type
    sf::RectangleShape basicButton = createButton(50, 500, 50, 50, basicTower.getColor());
    sf::RectangleShape advancedButton = createButton(0, 500, 50, 50, advancedTower.getColor());
    sf::RectangleShape ultimateButton = createButton(50, 550, 50, 50, ultimateTower.getColor());
    sf::RectangleShape cashButton = createButton(100, 550, 50, 50, cashTower.getColor());

    sf::Text cost1 = createText(52, 502, std::to_string(30) + "$", font, 20, sf::Color::Black);
    sf::Text cost2 = createText(2, 502, std::to_string(50) + "$", font, 20, sf::Color::Black);
    sf::Text cost3 = createText(52, 552, std::to_string(120) + "$", font, 20, sf::Color::Black);
    sf::Text cost4 = createText(102, 552, std::to_string(300) + "$", font, 20, sf::Color::White);

    // Draw the buttons
    window.draw(basicButton);
    window.draw(advancedButton);
    window.draw(ultimateButton);
    window.draw(cashButton);

    // Draw the cost text
    window.draw(cost1);
    window.draw(cost2);
    window.draw(cost3);
    window.draw(cost4);


    // Check if a button was clicked
    if (event.type == sf::Event::MouseButtonPressed &&
        event.mouseButton.button == sf::Mouse::Left) {
        if (basicButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
            selectedTowerType = &basicTower;
        } else if (advancedButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
            selectedTowerType = &advancedTower;
        } else if (ultimateButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
            selectedTowerType = &ultimateTower;
        } else if (cashButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
            selectedTowerType = &cashTower;
        }
    }

    // Check if a tile was clicked
    if (selectedTowerType != nullptr && event.mouseButton.button == sf::Mouse::Right) {
        sf::Vector2i mousePos = sf::Mouse::getPosition(window);
        sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
        Tile& closestTile = findClosestTile(tiles, rightPosition);


        if (closestTile.getColor() == sf::Color::Green) {

            if (money < selectedTowerType->getCost()) {
                std::cout « "Not enough money" « std::endl;

            }
            else{

            money -= selectedTowerType->getCost();

            addTower(window, closestTile, *selectedTowerType);


            selectedTowerType = nullptr;
            }
        }
    }
}

void drawTowers(sf::RenderWindow &window, int &money){
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distrib(0, 20);

    for (int i=0; i<towers.size();i++) {
        window.draw(towers[i].getShape());
        if (towers[i].attackEnemy(enemies)==1) {
            towers.erase(towers.begin() + i);
            std::cout « "Removing successful" « std::endl;
        }
        if (towers[i].attackEnemy(enemies)==2 && distrib(gen) == 1) {
            money += 1;
        }
    }
}


void onlyDrawTowers(sf::RenderWindow &window){
    for (int i=0; i<towers.size();i++){
        window.draw(towers[i].getShape());
        window.draw(towers[i].getAttackShape());
    }
}


void mainMenu(sf::RenderWindow &window, int difficulty) {
    window.clear();
        sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }

    std::vector<std::string> options = {"Easy", "Medium", "Hard", "Very Hard", "Insane"};
    std::vector<sf::RectangleShape> buttons;
    std::vector<sf::Text> texts;

    for (int i = 0; i < options.size(); i++) {
        buttons.push_back(createButton(300, 100 + i * 60, 200, 50, sf::Color::Blue));
        texts.push_back(createText(350, 110 + i * 60, options[i], font, 24,sf::Color::White));
    }
    //Create Tutorial button and text
    buttons.push_back(createButton(300, 40, 200, 50, sf::Color::Green));
    texts.push_back(createText(350, 50, "Tutorial", font, 24,sf::Color::Black));

    //Create Play button and text
    buttons.push_back(createButton(300, 400, 200, 50, sf::Color::Green));
    texts.push_back(createText(350, 410, "Play", font, 24,sf::Color::Black));

    //Create Exit button and text
    buttons.push_back(createButton(300, 460, 200, 50, sf::Color::Red));
    texts.push_back(createText(350, 470, "Exit", font, 24,sf::Color::White));


    for (int i = 0; i < buttons.size(); i++) {
        if(difficulty == i+1){
            buttons[i].setFillColor(sf::Color::Yellow);
            texts[i].setFillColor(sf::Color::Black);
        }
            window.draw(buttons[i]);
            window.draw(texts[i]);
    }


}

void drawMoney(sf::RenderWindow &window, int money) {
        sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }

    std::string moneyString = std::to_string(money) + "$";

    sf::Text moneyText = createText(680, -10, moneyString, font, 50, sf::Color::Yellow);

    if (money > 99 && money < 1000) {
        sf::Text moneyText = createText(660, -10, moneyString, font, 50, sf::Color::Yellow);
    }
    else if (money >= 1000) {
        sf::Text moneyText = createText(635, -10, moneyString, font, 50, sf::Color::Yellow);
    }

    window.draw(moneyText);
}
void endScreen(sf::RenderWindow &window) {
    sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }

    sf::Text text = createText(250, 200, "You lost!", font, 50,sf::Color::Red);
    sf::Text text2 = createText(20, 300, "Try with easier difficulty ;)", font, 45,sf::Color::Red);
    sf::Text text3 = createText(150, 400, "Click to try again", font, 45,sf::Color::Red);
    window.clear();
    window.draw(text);
    window.draw(text2);
    window.draw(text3);
}
void deleteTower(sf::Event event, sf::RenderWindow &window,int &money){
    if (event.type == sf::Event::MouseButtonPressed &&
        event.mouseButton.button == sf::Mouse::Left) {
        sf::Vector2i mousePos = sf::Mouse::getPosition(window);
        sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
        Tile& closestTile = findClosestTile(tiles, rightPosition);
            for (int i = 0; i < towers.size(); i++) {
                if (towers[i].getPosition() == closestTile.getPosition()) {
                    money += towers[i].getType().getCost()/2;
                    towers.erase(towers.begin() + i);
                }
            }
    }
}

void tutorial(sf::RenderWindow &window){
    window.clear();
    sf::Text tutorialText;
    sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }
    tutorialText.setFont(font);
    tutorialText.setString("Welcome to the game! Here’s how to play:\n\n"
                       "1. Start by selecting a difficulty level.\n"
                       "2. Press the ’Play’ button to start the game.\n"
                       "3. The game consists of two phases: Building and Attacking.\n"
                       "4. During the Building phase, you can build or remove towers.\n"
                       "5. To build a tower, first left-click a colored square to select the type of tower. \n   Then, right-click on a green tile to place the tower. \n   Make sure you have enough money to buy the tower!\n"
                       "6. You can remove a tower by left-clicking it during the Building phase. \n You’ll get some money back when you do this.\n   Range of each tower is visible in building stage.\n"
                       "7. When you’re ready, press the black cube to switch to the Attacking phase.\n Once no enemies are left standing\n   building phase starts again, Good luck! \n\n"
                       "Towers\n"
                       "1. Red Tower: Cost: 30$, Damage: 20, Range: 100, Attack Speed: 50,\n   special skill: none\n"
                       "2. Blue Tower: Cost: 50$, Damage: 30, Range: 100, Attack Speed: 60,\n special skill: slows enemies\n"
                       "3. Yellow Tower: Cost: 120$, Damage: 40, Range: 250, Attack Speed: 70,\n special skill: Huge range and Damage \n"
                       "4. Black Tower: Cost: 300$, Damage: 0, Range: 100, Attack Speed: 50,\n special skill: Gives you money\n\n"
                       "Enemies\n"
                       "1. Red Enemy: Health: 300, Speed: 2.5, Reward: 1\n   special skill: none\n"
                       "2. Cyan Enemy: Health: 100, Speed: 5, Reward: 2\n   special skill: very fast\n"
                       "3. Black Enemy: Health: 2500, Speed: 1, Reward: 5\n   special skill: very tanky\n"
                       "4. Yellow Enemy: Health: 2500, Speed: 1, Reward: 0\n   special skill: can destroy red towers\n");
    tutorialText.setCharacterSize(15);
    tutorialText.setFillColor(sf::Color::White);
    tutorialText.setPosition(50, 5);


    sf::RectangleShape backButton = createButton(100, 540, 200, 50, sf::Color::Blue);
    sf::Text text = createText(150, 545, "Back", font, 30,sf::Color::White);

    window.draw(backButton);
    window.draw(text);
    window.draw(tutorialText);
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
        sf::Vector2i mousePos = sf::Mouse::getPosition(window);
        if(backButton.getGlobalBounds().contains(mousePos.x,mousePos.y)){
            toMain = true;
        }
    }
}



void drawWave(sf::RenderWindow &window,int &gameLevel){
    sf::Font font;
    if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
        std::cout « "Could not load font" « std::endl;
    }

    std::string waveString = "Wave: " + std::to_string(gameLevel-1);

    sf::Text text = createText(290, -10, waveString, font, 50,sf::Color::Black);
    window.draw(text);
}
```

6.6 src/main.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include <cstdlib>
#include <ctime>
#include "Graphics/graphicFunctions.cpp"
#include "Objects/enemies.h"
#include "gameEngine.hpp"
#include <iostream>
#include <vector>
```

Include dependency graph for main.cpp:

Enumerations

- enum class GameState {
  MainMenu , Building , Attacking , EndScreen ,
  Tutorial }

Functions

- void moveEnemies (UniversalClock &clock, sf::RenderWindow &window, std::vector< Enemy > &stored_↩
  enemies, float delayTime, int &Money)
- int main ()

Variables

- UniversalClock clock1

6.6.1 Enumeration Type Documentation

6.6.1.1 GameState

```
enum class GameState [strong]
```

Enumerator

MainMenu	
Building	
Attacking	
EndScreen	
Tutorial	

6.6.2 Function Documentation

6.6.2.1 main()

```
int main ( )
```

6.6.2.2 moveEnemies()

```
void moveEnemies (
            UniversalClock & clock,
            sf::RenderWindow & window,
            std::vector< Enemy > & stored_enemies,
            float delayTime,
            int & Money )
```

Moves the enemies on the screen based on the given clock, window, and delay time. Also updates the money based on the enemies killed.

Parameters

clock	The UniversalClock object used to track the delay time.
window	The sf::RenderWindow object used to draw the enemies.
stored_enemies	The vector of Enemy objects representing the enemies on the screen.
delayTime	The delay time in seconds between enemy movements.
Money	The reference to the money variable to update based on enemies killed.

6.6.3 Variable Documentation

6.6.3.1 clock1

```
UniversalClock clock1
```

6.7 src/Objects/enemies.cpp File Reference

```
#include <iostream>
#include "enemies.h"
#include "EnemyType.h"
#include <random>
```

Include dependency graph for enemies.cpp:

6.8 src/Objects/enemies.h File Reference

```
#include <vector>
#include <string>
#include <SFML/Graphics.hpp>
#include "EnemyType.h"
#include "../gameEngine.hpp"
#include <iostream>
```

Include dependency graph for enemies.h: This graph shows which files directly or indirectly include this file:

Classes

- class Enemy

  Enemy class.

6.9 enemies.h

Go to the documentation of this file.

```
#ifndef TOWER_DEFENCE_2_ENEMY_H
#define TOWER_DEFENCE_2_ENEMY_H

#include <vector>
#include <string>
#include <SFML/Graphics.hpp>
#include "EnemyType.h"
#include "../gameEngine.hpp"
#include <iostream>

class EnemyType;


class Enemy  {
public:
    Enemy(sf::Vector2f& position, double radius, int health, double speed, float x, float y,
sf::Color& color,int points);
    sf::CircleShape& getShape();
    sf::Vector2f& getPosition();


    void move(float x_dir, float y_dir);
    void moveEnemy(double timeStep, sf::RenderWindow &window);

    int getRoute();
    int getSpeed();
    int getXcoord();
    int getYcoord();

    void addY(int b);
    void addX(int a);

    void lowerHealth(int h);

    bool hasPassed();
    bool isDead();

    void getHit(int damage){
        health -= damage;
    }

    int getHealth(){
        return this->health;
    }
    int getPoints(){
        return this->points;
    }
    void reduceSpeed(){
        if(this->shape.getFillColor() == sf::Color::Cyan){
            speed = 2;
        }
        else if(this->shape.getFillColor() == sf::Color::Black){
            speed = 1;
        }
        else if(this->shape.getFillColor() == sf::Color::Red){
            speed = 1;
        }
    }
    // destructor
    ~Enemy(){

    }
    private:
        sf::CircleShape shape;
        sf::Vector2f position;
        int x;
        int y;
        float speed;
        int health;
        int points;
        int route;
};

#endif
```

6.10 src/Objects/EnemyType.h File Reference

`#include <SFML/Graphics.hpp>`

Include dependency graph for EnemyType.h:

6.11 EnemyType.h

Go to the documentation of this file.

```
// EnemyType.h
#ifndef TOWER_DEFENCE_2_ENEMYTYPE_H
#define TOWER_DEFENCE_2_ENEMYTYPE_H

#include <SFML/Graphics.hpp>

class Enemy;

class EnemyType {
public:
    virtual Enemy createEnemy(sf::Vector2f& position, float x, float y) const = 0;

    virtual ~EnemyType() = default;
};

#endif
```

6.12 src/Objects/EnemyTypeA.cpp File Reference

```
#include "EnemyTypeA.h"
#include "enemies.h"
#include <SFML/Graphics.hpp>
```

Include dependency graph for EnemyTypeA.cpp:

6.13 src/Objects/EnemyTypeA.h File Reference

`#include "EnemyType.h"`

Include dependency graph for EnemyTypeA.h: This graph shows which files directly or indirectly include this file:

Classes

- class EnemyTypeA

6.14 EnemyTypeA.h

Go to the documentation of this file.

```
// EnemyTypeA.h
#ifndef TOWER_DEFENCE_2_ENEMYTYPEA_H
#define TOWER_DEFENCE_2_ENEMYTYPEA_H

#include "EnemyType.h"

class EnemyTypeA : public EnemyType {
public:
    Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
};

#endif
```

6.15 src/Objects/EnemyTypeB.cpp File Reference

```
#include "EnemyTypeB.h"
#include "enemies.h"
#include <SFML/Graphics.hpp>
```

Include dependency graph for EnemyTypeB.cpp:

6.16 src/Objects/EnemyTypeB.h File Reference

```
#include "EnemyType.h"
```

Include dependency graph for EnemyTypeB.h: This graph shows which files directly or indirectly include this file:

Classes

- class EnemyTypeB

  EnemyTypeB.h.

6.17 EnemyTypeB.h

Go to the documentation of this file.

```

#ifndef TOWER_DEFENCE_2_ENEMYTYPEB_H
#define TOWER_DEFENCE_2_ENEMYTYPEB_H

#include "EnemyType.h"

class EnemyTypeB : public EnemyType {
public:
    Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
};

#endif
```

6.18 src/Objects/EnemyTypeC.cpp File Reference

```
#include "EnemyTypeC.h"
#include "enemies.h"
#include <SFML/Graphics.hpp>
```

Include dependency graph for EnemyTypeC.cpp:

6.19 src/Objects/EnemyTypeC.h File Reference

```
#include "EnemyType.h"
```

Include dependency graph for EnemyTypeC.h: This graph shows which files directly or indirectly include this file:

Classes

- class EnemyTypeC

  EnemyTypeC.h.

6.20 EnemyTypeC.h

Go to the documentation of this file.

```

#ifndef TOWER_DEFENCE_2_ENEMYTYPEC_H
#define TOWER_DEFENCE_2_ENEMYTYPEC_H

#include "EnemyType.h"

class EnemyTypeC : public EnemyType {
public:
    Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
};

#endif
```

6.21 src/Objects/EnemyTypeD.cpp File Reference

```
#include "EnemyTypeD.h"
#include "enemies.h"
#include <SFML/Graphics.hpp>
```

Include dependency graph for EnemyTypeD.cpp:

6.22 src/Objects/EnemyTypeD.h File Reference

```
#include "EnemyType.h"
```

Include dependency graph for EnemyTypeD.h: This graph shows which files directly or indirectly include this file:

Classes

- class EnemyTypeD

  EnemyTypeD.h.

6.23 EnemyTypeD.h

Go to the documentation of this file.

```

#ifndef TOWER_DEFENCE_2_ENEMYTYPED_H
#define TOWER_DEFENCE_2_ENEMYTYPED_H

#include "EnemyType.h"

class EnemyTypeD : public EnemyType {
public:
    Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
};

#endif
```

6.24 src/Objects/tower.cpp File Reference

```
#include "tower.h"
#include <iostream>
```

Include dependency graph for tower.cpp:

6.25 src/Objects/tower.h File Reference

```
#include <SFML/Graphics.hpp>
#include "enemies.h"
#include "../gameEngine.hpp"
```

Include dependency graph for tower.h: This graph shows which files directly or indirectly include this file:

Classes

- class TowerType

  Tower class.
- class Tower

6.26 tower.h

Go to the documentation of this file.

```
#ifndef TOWER_H
#define TOWER_H

#include <SFML/Graphics.hpp>
#include "enemies.h"
#include "../gameEngine.hpp"




class TowerType {
public:
    TowerType(double radius, int damage, double attack_range, double attack_speed, const sf::Color&
color, int cost);

    double getRadius() const;

    int getDamage() const;

    double getAttackRange() const;

    double getAttackSpeed() const;

    const sf::Color& getColor() const;

    int getCost() const {return cost;}

    ~TowerType() {
    }

private:
    double radius;
    int damage;
    double attack_range;
    double attack_speed;
    sf::Color color;
    int cost;
};
class Tower  {
public:
    Tower(const sf::Vector2f& position, const TowerType& type);
    sf::ConvexShape& getShape();
    double getAttack_range() const;
    int attackEnemy(std::vector<Enemy> &enemies);
    sf::CircleShape getAttackShape();
    sf::Vector2f getPosition();
    // function to add clock to vector of clocks
    void addClock(UniversalClock &clock);
    TowerType getType() const {return type;}
    ~Tower() {
    }

private:
    sf::ConvexShape shape;
    TowerType type;
    sf::CircleShape attackShape;
    // list of universal clocks
    std::vector<UniversalClock> clocks;

};



#endif
```

6.27 src/tiles.cpp File Reference

```
#include <SFML/Graphics.hpp>
#include <cmath>
```

Include dependency graph for tiles.cpp: This graph shows which files directly or indirectly include this file:

Classes

- class Tile

  Represents a tile in the game.

Functions

- Tile & findClosestTile (std::vector< Tile > &tiles, const sf::Vector2f &position)

6.27.1 Function Documentation

6.27.1.1 findClosestTile()

```
Tile & findClosestTile (
            std::vector< Tile > & tiles,
            const sf::Vector2f & position )
```

Finds the closest Tile to the given position from the provided vector of Tiles.

Parameters

tiles	The vector of Tiles to search from.
position	The position to find the closest Tile to.

Returns

A reference to the closest Tile.

6.28 tiles.cpp

Go to the documentation of this file.

```cpp
#include <SFML/Graphics.hpp>
#include <cmath>
//Tile class for tile objects
class Tile {
public:
    Tile(const sf::Vector2f& position, const sf::Color& color, int tileSize)
    :position(position),color(color),tileSize(tileSize) {
        shape.setSize(sf::Vector2f(tileSize, tileSize));
        shape.setPosition(position);
        shape.setFillColor(color);
        if(color == sf::Color::Green){
            shape.setOutlineColor(sf::Color::Black);
        }
        else{
            shape.setOutlineColor(color);
        }
        shape.setOutlineThickness(0.5);
    }

    sf::RectangleShape& getShape() {
        return shape;
    }

    sf::Vector2f& getPosition() {
        return position;
    }

    const sf::Color& getColor() const{
        return color;
    }

private:
    sf::RectangleShape shape;
    sf::Vector2f position;
    sf::Color color;
    int tileSize;
};

Tile& findClosestTile(std::vector<Tile>& tiles, const sf::Vector2f& position) {
    Tile* closestTile = nullptr;
    float minDistance = std::numeric_limits<float>::max();

    for (Tile& tile : tiles) {
        sf::Vector2f tileCenter = tile.getPosition()+sf::Vector2f(25,25);
        float distance = std::hypot(position.x - tileCenter.x, position.y - tileCenter.y);

        if (distance < minDistance) {
            minDistance = distance;
            closestTile = const_cast<Tile*>(&tile);
        }
    }

    return *closestTile;
}
```

Index