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.

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 )
```

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]
```

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

Public Member Functions

- Tile (const sf::Vector2f &position, const sf::Color &color, int tileSize)
- sf::RectangleShape & getShape ()
- sf::Vector2f & getPosition ()
- const sf::Color & getColor () const

5.7.1 Constructor & Destructor Documentation

5.7.1.1 Tile()

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

5.7.2 Member Function Documentation

5.7.2.1 getColor()

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

5.7.2.2 getPosition()

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

5.7.2.3 getShape()

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

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.cpp File Reference

```
#include "gameEngine.hpp"
```
Include dependency graph for gameEngine.cpp:

Macros

- #define GAME_ENGINE_HPP

6.2.1 Macro Definition Documentation

6.2.1.1 GAME_ENGINE_HPP

```
#define GAME_ENGINE_HPP
```

6.3 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.

6.4 gameEngine.hpp

Go to the documentation of this file.

```
00001 #ifndef GAME_ENGINE_HPP
00002 #define GAME_ENGINE_HPP
00003 #include <SFML/Graphics.hpp>
00004 #include "Objects/enemies.h"
00005
00006
00007
00011 class UniversalClock {
00012 public:
00016     UniversalClock() : clock() {}
00017
00023     bool isDelayFinished(float delayTime) {
00024         return clock.getElapsedTime().asMilliseconds() >= delayTime;
00025     }
00026
00030     void restartClock() {
00031         clock.restart();
00032     }
00033
00034 private:
00035     sf::Clock clock;
00036 };
00037 #endif
00038 // void delayedFunction(UniversalClock &clock, float delayTime,Enemy &enemy1, sf::RenderWindow
      &window) {
00039 //     if (clock.isDelayFinished(delayTime)) {
00040 //         enemy1.moveEnemy(0.1,window);
00041 //         clock.restartClock();
00042
00043 //     }
00044 // }
```

6.5 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)
- 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)
- 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)
- void endScreen (sf::RenderWindow &window)
- void deleteTower (sf::Event event, sf::RenderWindow &window, int &money)
- 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

6.5.1 Function Documentation

6.5.1.1 addEnemy()

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

6.5.1.2 addTower()

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

6.5.1.3 createButton()

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

6.5.1.4 createText()

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

6.5.1.5 deleteTower()

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

6.5.1.6 drawMoney()

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

6.5.1.7 drawTiles()

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

6.5.1.8 drawTowers()

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

6.5.1.9 drawWave()

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

6.5.1.10 endScreen()

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

6.5.1.11 mainMenu()

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

6.5.1.12 onlyDrawTowers()

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

6.5.1.13 placeTower()

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

6.5.1.14 tutorial()

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

6.5.2 Variable Documentation

6.5.2.1 enemies

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

6.5.2.2 exitButton

```
sf::RectangleShape exitButton
```

6.5.2.3 playButton

```
sf::RectangleShape playButton
```

6.5.2.4 selectedTowerType

```
TowerType* selectedTowerType = nullptr
```

6.5.2.5 tiles

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

6.5.2.6 toMain

```
bool toMain = false
```

6.5.2.7 towerPlacementMode

```
bool towerPlacementMode = false
```

6.5.2.8 towers

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

6.6 graphicFunctions.cpp

Go to the documentation of this file.

```
00001 #include <SFML/Graphics.hpp>
00002 #include <cstdlib>
00003 #include <ctime>
00004 #include <vector>
00005 #include <iostream>
00006 #include <fstream>
00007 #include <random>
00008 #include "../Objects/tower.h"
00009 #include "../Objects/enemies.h"
00010 #include "../tiles.cpp"
00011 #include "../Objects/EnemyTypeA.h"
00012 #include "../Objects/EnemyTypeB.h"
00013 #include "../Objects/EnemyTypeC.h"
00014 #include "../Objects/EnemyTypeD.h"
00015
00016 std::vector<Tile> tiles;
00017 std::vector<Tower> towers;
00018 std::vector<Enemy> enemies;
00019 sf::RectangleShape playButton;
00020 sf::RectangleShape exitButton;
00021 bool towerPlacementMode = false;
00022 TowerType* selectedTowerType = nullptr;
00023 bool toMain = false;
00024 //function to draw all the tiles from hardcoded map
00025 void drawTiles(sf::RenderWindow &window, const int tileSize, const int windowWidth, const int
      windowHeight,int difficulty) {
00026     //Count how many tiles we can fit in map
00027     const int mapWidth = windowWidth / tileSize;
00028     const int mapHeight = windowHeight / tileSize;
00029
00030
00031     // Define a hardcoded map, 0 for water(blue), 1 for grass(green) and other for path(white)
00032     int map[16][12];
00033     std::ifstream mapFile;
00034     if(difficulty > 3){
00035         mapFile.open("src/assets/map2.txt");
00036     }
00037     else{
00038         mapFile.open("src/assets/map1.txt");
00039     }
00040
00041     if (!mapFile) {
00042         std::cerr « "Unable to open map file";
00043         // handle error
00044     }
00045
00046     for (int i = 0; i < 16; ++i) {
00047         for (int j = 0; j < 12; ++j) {
00048             char ch;
00049             if (!(mapFile » ch)) {
00050                 //std::cerr « "Error reading map file";
00051                 // handle error
00052             }
00053             map[i][j] = ch - '0';  // convert char to int
00054         }
```

```
00055     }
00056
00057     mapFile.close();
00058
00059     //iterate through all the tiles
00060     for (int x = 0; x < mapWidth; x++) {
00061         for (int y = 0; y < mapHeight; y++) {
00062             //calculate position
00063             sf::Vector2f tilePosition(x * tileSize, y * tileSize);
00064
00065             // Create a Tile object with the specified color, and tileSize
00066             int tileType = map[x][y];
00067             sf::Color tileColor;
00068             if (tileType == 0) {
00069                 tileColor = sf::Color::Blue;
00070             } else if (tileType == 1) {
00071                 tileColor = sf::Color::Green;
00072             } else {
00073                 tileColor = sf::Color::White;
00074             }
00075             //create tile objects for each tile
00076             Tile tile(tilePosition, tileColor, tileSize);
00077
00078             tiles.push_back(tile);
00079             // Draw the tile's shape
00080             window.draw(tile.getShape());
00081         }
00082     }
00083 }
00084 sf::RectangleShape createButton(float x, float y, float width, float height, sf::Color color) {
00085     sf::RectangleShape button(sf::Vector2f(width, height));
00086     button.setPosition(x, y);
00087     button.setFillColor(color);
00088     return button;
00089 }
00090
00091 // Function to create a text
00092 sf::Text createText(float x, float y, std::string content, sf::Font& font, unsigned int size,sf::Color
      color) {
00093     sf::Text text;
00094     text.setFont(font);
00095     text.setFillColor(color);
00096     text.setString(content);
00097     text.setCharacterSize(size);
00098     text.setPosition(x, y);
00099     return text;
00100 }
00101
00102 void addTower(sf::RenderWindow &window, Tile tile, TowerType type){
00103     Tower tower(tile.getPosition(),type);
00104         towers.push_back(tower);
00105 }
00106
00107 void addEnemy(sf::RenderWindow &window, int tileSize, int x, int y, int gameLevel, int difficulty){
00108
00109     int iterator = gameLevel * difficulty;
00110
00111     if (gameLevel <= 2) {
00112         for (int j = 1; j < 3*iterator; j++) {
00113
00114             sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
00115             EnemyTypeA enemyTypeA;
00116             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
00117         }
00118     }
00119     else if (gameLevel > 2 && gameLevel <= 4) {
00120         for (int j = 1; j < 3*iterator; j++) {
00121             sf::Vector2f tileStartPosition_B((-j-5)*tileSize+7, y * tileSize+7);
00122             EnemyTypeB enemyTypeB;
00123             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
      (-j-5)*tileSize,y*tileSize));
00124         }
00125         for (int j = 1; j < 3*iterator; j++) {
00126             sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
00127             EnemyTypeA enemyTypeA;
00128             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
00129         }
00130     }
00131     else if (gameLevel > 4 && gameLevel <= 6) {
00132         sf::Vector2f tileStartPosition_D((x-10.5)*tileSize+1, y * tileSize+1);
00133         EnemyTypeD enemyTypeD;
00134         enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-10.5)*tileSize,y*tileSize));
00135
00136         for (int j = 1; j < 4*iterator; j++) {
00137             sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
00138             EnemyTypeB enemyTypeB;
00139             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
```

```
    (-j-15)*tileSize,y*tileSize));
00140         }
00141         for (int j = 1; j < 2*iterator; j++){
00142             sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
00143             EnemyTypeA enemyTypeA;
00144             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A,
    (-j-6)*tileSize,y*tileSize));
00145         }
00146         for (int j = 1; j < 1*iterator; j++){
00147             sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
00148             EnemyTypeC enemyTypeC;
00149             enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C,
    (-j-2)*tileSize,y*tileSize));
00150         }
00151     }
00152     else if (gameLevel > 6) {
00153         sf::Vector2f tileStartPosition_D((x-11.5)*tileSize+1, y * tileSize+1);
00154         EnemyTypeD enemyTypeD;
00155         enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-11.5)*tileSize,y*tileSize));
00156 
00157         for (int j = 1; j < 8*iterator; j++) {
00158             sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
00159             EnemyTypeB enemyTypeB;
00160             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
    (-j-15)*tileSize,y*tileSize));
00161         }
00162         for (int j = 1; j < 4*iterator; j++){
00163             sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
00164             EnemyTypeA enemyTypeA;
00165             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A,
    (-j-6)*tileSize,y*tileSize));
00166         }
00167         for (int j = 1; j < 2*iterator; j++){
00168             sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
00169             EnemyTypeC enemyTypeC;
00170             enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C,
    (-j-2)*tileSize,y*tileSize));
00171         }
00172     }
00173 
00174 
00175 }
00176 
00177 
00178 
00179 void placeTower(sf::Event event, sf::RenderWindow &window, int &money){
00180 
00181     sf::Font font;
00182     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00183         std::cout « "Could not load font" « std::endl;
00184     }
00185 
00186     //Define the tower types
00187     TowerType basicTower(30.0, 20, 100, 50.0, sf::Color::Red,30);
00188     TowerType advancedTower(40.0, 30, 100, 60.0, sf::Color::Blue,50);
00189     TowerType ultimateTower(50.0, 40, 250, 70.0, sf::Color::Yellow,120);
00190     TowerType cashTower(0,0,100,50,sf::Color::Black,300);
00191 
00192     // Create the buttons for each tower type
00193     sf::RectangleShape basicButton = createButton(50, 500, 50, 50, basicTower.getColor());
00194     sf::RectangleShape advancedButton = createButton(0, 500, 50, 50, advancedTower.getColor());
00195     sf::RectangleShape ultimateButton = createButton(50, 550, 50, 50, ultimateTower.getColor());
00196     sf::RectangleShape cashButton = createButton(100, 550, 50, 50, cashTower.getColor());
00197 
00198     sf::Text cost1 = createText(52, 502, std::to_string(30) + "$", font, 20, sf::Color::Black);
00199     sf::Text cost2 = createText(2, 502, std::to_string(50) + "$", font, 20, sf::Color::Black);
00200     sf::Text cost3 = createText(52, 552, std::to_string(120) + "$", font, 20, sf::Color::Black);
00201     sf::Text cost4 = createText(102, 552, std::to_string(300) + "$", font, 20, sf::Color::White);
00202 
00203     // Draw the buttons
00204     window.draw(basicButton);
00205     window.draw(advancedButton);
00206     window.draw(ultimateButton);
00207     window.draw(cashButton);
00208 
00209     // Draw the cost text
00210     window.draw(cost1);
00211     window.draw(cost2);
00212     window.draw(cost3);
00213     window.draw(cost4);
00214 
00215 
00216     // Check if a button was clicked
00217     if (event.type == sf::Event::MouseButtonPressed &&
00218         event.mouseButton.button == sf::Mouse::Left) {
00219         if (basicButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
00220             selectedTowerType = &basicTower;
```

```
00221         } else if (advancedButton.getGlobalBounds().contains(event.mouseButton.x,
      event.mouseButton.y)) {
00222             selectedTowerType = &advancedTower;
00223         } else if (ultimateButton.getGlobalBounds().contains(event.mouseButton.x,
      event.mouseButton.y)) {
00224             selectedTowerType = &ultimateTower;
00225         } else if (cashButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
00226             selectedTowerType = &cashTower;
00227         }
00228     }
00229
00230     // Check if a tile was clicked
00231     if (selectedTowerType != nullptr && event.mouseButton.button == sf::Mouse::Right) {
00232         sf::Vector2i mousePos = sf::Mouse::getPosition(window);
00233         sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
00234         Tile& closestTile = findClosestTile(tiles, rightPosition);
00235
00236
00237         if (closestTile.getColor() == sf::Color::Green) {
00238
00239             if (money < selectedTowerType->getCost()) {
00240                 std::cout « "Not enough money" « std::endl;
00241
00242             }
00243             else{
00244
00245             money -= selectedTowerType->getCost();
00246
00247             addTower(window, closestTile, *selectedTowerType);
00248
00249
00250             selectedTowerType = nullptr;
00251             }
00252         }
00253     }
00254 }
00255
00256 void drawTowers(sf::RenderWindow &window, int &money){
00257     std::random_device rd;
00258     std::mt19937 gen(rd());
00259     std::uniform_int_distribution<> distrib(0, 20);
00260
00261     for (int i=0; i<towers.size();i++) {
00262         window.draw(towers[i].getShape());
00263         if (towers[i].attackEnemy(enemies)==1) {
00264             towers.erase(towers.begin() + i);
00265             std::cout « "Removing successful" « std::endl;
00266         }
00267         if (towers[i].attackEnemy(enemies)==2 && distrib(gen) == 1) {
00268             money += 1;
00269         }
00270     }
00271 }
00272
00273
00274 void onlyDrawTowers(sf::RenderWindow &window){
00275     for (int i=0; i<towers.size();i++){
00276         window.draw(towers[i].getShape());
00277         window.draw(towers[i].getAttackShape());
00278     }
00279 }
00280
00281
00282 void mainMenu(sf::RenderWindow &window, int difficulty) {
00283     window.clear();
00284         sf::Font font;
00285     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00286         std::cout « "Could not load font" « std::endl;
00287     }
00288
00289     std::vector<std::string> options = {"Easy", "Medium", "Hard", "Very Hard", "Insane"};
00290     std::vector<sf::RectangleShape> buttons;
00291     std::vector<sf::Text> texts;
00292
00293     for (int i = 0; i < options.size(); i++) {
00294         buttons.push_back(createButton(300, 100 + i * 60, 200, 50, sf::Color::Blue));
00295         texts.push_back(createText(350, 110 + i * 60, options[i], font, 24,sf::Color::White));
00296     }
00297     //Create Tutorial button and text
00298     buttons.push_back(createButton(300, 40, 200, 50, sf::Color::Green));
00299     texts.push_back(createText(350, 50, "Tutorial", font, 24,sf::Color::Black));
00300
00301     //Create Play button and text
00302     buttons.push_back(createButton(300, 400, 200, 50, sf::Color::Green));
00303     texts.push_back(createText(350, 410, "Play", font, 24,sf::Color::Black));
00304
00305     //Create Exit button and text
```

```
00306     buttons.push_back(createButton(300, 460, 200, 50, sf::Color::Red));
00307     texts.push_back(createText(350, 470, "Exit", font, 24,sf::Color::White));
00308
00309
00310     for (int i = 0; i < buttons.size(); i++) {
00311         if(difficulty == i+1){
00312             buttons[i].setFillColor(sf::Color::Yellow);
00313             texts[i].setFillColor(sf::Color::Black);
00314         }
00315             window.draw(buttons[i]);
00316             window.draw(texts[i]);
00317     }
00318
00319
00320 }
00321
00322 void drawMoney(sf::RenderWindow &window, int money) {
00323         sf::Font font;
00324     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00325         std::cout « "Could not load font" « std::endl;
00326     }
00327
00328     std::string moneyString = std::to_string(money) + "$";
00329
00330     sf::Text moneyText = createText(680, -10, moneyString, font, 50, sf::Color::Yellow);
00331
00332     if (money > 99 && money < 1000) {
00333         sf::Text moneyText = createText(660, -10, moneyString, font, 50, sf::Color::Yellow);
00334     }
00335     else if (money >= 1000) {
00336         sf::Text moneyText = createText(635, -10, moneyString, font, 50, sf::Color::Yellow);
00337     }
00338
00339     window.draw(moneyText);
00340 }
00341 void endScreen(sf::RenderWindow &window) {
00342     sf::Font font;
00343     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00344         std::cout « "Could not load font" « std::endl;
00345     }
00346
00347     sf::Text text = createText(250, 200, "You lost!", font, 50,sf::Color::Red);
00348     sf::Text text2 = createText(20, 300, "Try with easier difficulty ;)", font, 45,sf::Color::Red);
00349     sf::Text text3 = createText(150, 400, "Click to try again", font, 45,sf::Color::Red);
00350     window.clear();
00351     window.draw(text);
00352     window.draw(text2);
00353     window.draw(text3);
00354 }
00355 void deleteTower(sf::Event event, sf::RenderWindow &window,int &money){
00356     if (event.type == sf::Event::MouseButtonPressed &&
00357         event.mouseButton.button == sf::Mouse::Left) {
00358         sf::Vector2i mousePos = sf::Mouse::getPosition(window);
00359         sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
00360         Tile& closestTile = findClosestTile(tiles, rightPosition);
00361             for (int i = 0; i < towers.size(); i++) {
00362                 if (towers[i].getPosition() == closestTile.getPosition()) {
00363                     money += towers[i].getType().getCost()/2;
00364                     towers.erase(towers.begin() + i);
00365                 }
00366             }
00367     }
00368 }
00369
00370 void tutorial(sf::RenderWindow &window){
00371     window.clear();
00372     sf::Text tutorialText;
00373     sf::Font font;
00374     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00375         std::cout « "Could not load font" « std::endl;
00376     }
00377     tutorialText.setFont(font);
00378     tutorialText.setString("Welcome to the game! Here's how to play:\n\n"
00379                         "1. Start by selecting a difficulty level.\n"
00380                         "2. Press the 'Play' button to start the game.\n"
00381                         "3. The game consists of two phases: Building and Attacking.\n"
00382                         "4. During the Building phase, you can build or remove towers.\n"
00383                         "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"
00384                         "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"
00385                         "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"
00386                         "Towers\n"
00387                         "1. Red Tower: Cost: 30$, Damage: 20, Range: 100, Attack Speed: 50,\n   special
      skill: none\n"
```

```
00388                       "2. Blue Tower: Cost: 50$, Damage: 30, Range: 100, Attack Speed: 60,\n
      special skill: slows enemies\n"
00389                       "3. Yellow Tower: Cost: 120$, Damage: 40, Range: 250, Attack Speed: 70,\n
      special skill: Huge range and Damage \n"
00390                       "4. Black Tower: Cost: 300$, Damage: 0, Range: 100, Attack Speed: 50,\n
      special skill: Gives you money\n\n"
00391                       "Enemies\n"
00392                       "1. Red Enemy: Health: 300, Speed: 2.5, Reward: 1\n   special skill: none\n"
00393                       "2. Cyan Enemy: Health: 100, Speed: 5, Reward: 2\n   special skill: very
      fast\n"
00394                       "3. Black Enemy: Health: 2500, Speed: 1, Reward: 5\n   special skill: very
      tanky\n"
00395                       "4. Yellow Enemy: Health: 2500, Speed: 1, Reward: 0\n   special skill: can
      destroy red towers\n");
00396     tutorialText.setCharacterSize(15);
00397     tutorialText.setFillColor(sf::Color::White);
00398     tutorialText.setPosition(50, 5);
00399 
00400 
00401     sf::RectangleShape backButton = createButton(100, 540, 200, 50, sf::Color::Blue);
00402     sf::Text text = createText(150, 545, "Back", font, 30,sf::Color::White);
00403 
00404     window.draw(backButton);
00405     window.draw(text);
00406     window.draw(tutorialText);
00407     if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
00408         sf::Vector2i mousePos = sf::Mouse::getPosition(window);
00409         if(backButton.getGlobalBounds().contains(mousePos.x,mousePos.y)){
00410             toMain = true;
00411         }
00412     }
00413 }
00414 
00415 
00416 
00417 void drawWave(sf::RenderWindow &window,int &gameLevel){
00418     sf::Font font;
00419     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00420         std::cout << "Could not load font" << std::endl;
00421     }
00422 
00423     std::string waveString = "Wave: " + std::to_string(gameLevel-1);
00424 
00425     sf::Text text = createText(290, -10, waveString, font, 50,sf::Color::Black);
00426     window.draw(text);
00427 }
```

6.7 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

6.7.1 Enumeration Type Documentation

6.7.1.1 GameState

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

Enumerator

MainMenu	
Building	
Attacking	
EndScreen	
Tutorial	

6.7.2 Function Documentation

6.7.2.1 main()

```
int main ( )
```

6.7.2.2 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.

6.7.3 Variable Documentation

6.7.3.1 clock1

```
UniversalClock clock1
```

6.8 src/Objects/enemies.cpp File Reference

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

Include dependency graph for enemies.cpp:

6.9 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.

6.10 enemies.h

Go to the documentation of this file.

```
00001 #ifndef TOWER_DEFENCE_2_ENEMY_H
00002 #define TOWER_DEFENCE_2_ENEMY_H
00003
00004 #include <vector>
00005 #include <string>
00006 #include <SFML/Graphics.hpp>
00007 #include "EnemyType.h"
00008 #include "../gameEngine.hpp"
00009 #include <iostream>
00010
00011 class EnemyType;
00012
00014 class Enemy  {
00015 public:
00027     Enemy(sf::Vector2f& position, double radius, int health, double speed, float x, float y,
      sf::Color& color,int points);
00028     sf::CircleShape& getShape();
00029     sf::Vector2f& getPosition();
00030
00031
00032     void move(float x_dir, float y_dir);
00033     void moveEnemy(double timeStep, sf::RenderWindow &window);
00034
00035     int getRoute();
```

```
00036     int getSpeed();
00037     int getXcoord();
00038     int getYcoord();
00039
00040     void addY(int b);
00041     void addX(int a);
00042
00043     void lowerHealth(int h);
00044
00045     bool hasPassed();
00046     bool isDead();
00047
00048     void getHit(int damage){
00049         health -= damage;
00050     }
00051
00052     int getHealth(){
00053         return this->health;
00054     }
00055     int getPoints(){
00056         return this->points;
00057     }
00058     void reduceSpeed(){
00059         if(this->shape.getFillColor() == sf::Color::Cyan){
00060             speed = 2;
00061         }
00062         else if(this->shape.getFillColor() == sf::Color::Black){
00063             speed = 1;
00064         }
00065         else if(this->shape.getFillColor() == sf::Color::Red){
00066             speed = 1;
00067         }
00068     }
00069     // destructor
00070     ~Enemy(){
00071
00072     }
00073     private:
00074         sf::CircleShape shape;
00075         sf::Vector2f position;
00076         int x;
00077         int y;
00078         float speed;
00079         int health;
00080         int points;
00081         int route;
00082 };
00083
00084 #endif
```

6.11 src/Objects/EnemyType.h File Reference

`#include <SFML/Graphics.hpp>`
Include dependency graph for EnemyType.h:

6.12 EnemyType.h

Go to the documentation of this file.

```
00001 // EnemyType.h
00002 #ifndef TOWER_DEFENCE_2_ENEMYTYPE_H
00003 #define TOWER_DEFENCE_2_ENEMYTYPE_H
00004
00005 #include <SFML/Graphics.hpp>
00006
00007 class Enemy;
00008
00012 class EnemyType {
00013 public:
00022     virtual Enemy createEnemy(sf::Vector2f& position, float x, float y) const = 0;
00023
00027     virtual ~EnemyType() = default;
00028 };
00029
00030 #endif
```

6.13 src/Objects/EnemyTypeA.cpp File Reference

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

Include dependency graph for EnemyTypeA.cpp:

6.14 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

6.15 EnemyTypeA.h

Go to the documentation of this file.

```
00001 // EnemyTypeA.h
00002 #ifndef TOWER_DEFENCE_2_ENEMYTYPEA_H
00003 #define TOWER_DEFENCE_2_ENEMYTYPEA_H
00004
00005 #include "EnemyType.h"
00006
00007 class EnemyTypeA : public EnemyType {
00008 public:
00009     Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
00010 };
00011
00012 #endif
```

6.16 src/Objects/EnemyTypeB.cpp File Reference

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

Include dependency graph for EnemyTypeB.cpp:

6.17 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.

6.18 EnemyTypeB.h

Go to the documentation of this file.

```
00001
00002 #ifndef TOWER_DEFENCE_2_ENEMYTYPEB_H
00003 #define TOWER_DEFENCE_2_ENEMYTYPEB_H
00004
00005 #include "EnemyType.h"
00006
00007 class EnemyTypeB : public EnemyType {
00008 public:
00009     Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
00010 };
00011
00012 #endif
```

6.19 src/Objects/EnemyTypeC.cpp File Reference

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

Include dependency graph for EnemyTypeC.cpp:

6.20 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.

6.21 EnemyTypeC.h

Go to the documentation of this file.

```
00001
00002 #ifndef TOWER_DEFENCE_2_ENEMYTYPEC_H
00003 #define TOWER_DEFENCE_2_ENEMYTYPEC_H
00004
00005 #include "EnemyType.h"
00006
00007 class EnemyTypeC : public EnemyType {
00008 public:
00009     Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
00010 };
00011
00012 #endif
```

6.22 src/Objects/EnemyTypeD.cpp File Reference

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

Include dependency graph for EnemyTypeD.cpp:

6.23 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.

6.24 EnemyTypeD.h

Go to the documentation of this file.

```
00001
00002 #ifndef TOWER_DEFENCE_2_ENEMYTYPED_H
00003 #define TOWER_DEFENCE_2_ENEMYTYPED_H
00004
00005 #include "EnemyType.h"
00006
00007 class EnemyTypeD : public EnemyType {
00008 public:
00009     Enemy createEnemy(sf::Vector2f& position, float x, float y) const override;
00010 };
00011
00012 #endif
```

6.25 src/Objects/tower.cpp File Reference

```
#include "tower.h"
#include <iostream>
```
Include dependency graph for tower.cpp:

6.26 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

6.27 tower.h

Go to the documentation of this file.

```
00001 #ifndef TOWER_H
00002 #define TOWER_H
00003
00004 #include <SFML/Graphics.hpp>
00005 #include "enemies.h"
00006 #include "../gameEngine.hpp"
00007
00008
00009
00011
00014 class TowerType {
00015 public:
00025     TowerType(double radius, int damage, double attack_range, double attack_speed, const sf::Color&
      color, int cost);
00026
00031     double getRadius() const;
00032
00037     int getDamage() const;
00038
00043     double getAttackRange() const;
00044
00049     double getAttackSpeed() const;
00050
00055     const sf::Color& getColor() const;
00056
00061     int getCost() const {return cost;}
00062
00066     ~TowerType() {
00067     }
00068
00069 private:
00070     double radius;
00071     int damage;
00072     double attack_range;
00073     double attack_speed;
00074     sf::Color color;
00075     int cost;
00076 };
00077 class Tower  {
00078 public:
00079     Tower(const sf::Vector2f& position, const TowerType& type);
00080     sf::ConvexShape& getShape();
00081     double getAttack_range() const;
00082     int attackEnemy(std::vector<Enemy> &enemies);
00083     sf::CircleShape getAttackShape();
00084     sf::Vector2f getPosition();
00085     // function to add clock to vector of clocks
00086     void addClock(UniversalClock &clock);
00087     TowerType getType() const {return type;}
00088     ~Tower() {
00089     }
00090
00091 private:
00092     sf::ConvexShape shape;
00093     TowerType type;
00094     sf::CircleShape attackShape;
00095     // list of universal clocks
00096     std::vector<UniversalClock> clocks;
00097
00098 };
00099
00100
00101
00102 #endif
```

6.28 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

Functions

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

6.28.1 Function Documentation

6.28.1.1 findClosestTile()

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

6.29 tiles.cpp

Go to the documentation of this file.

```
00001 #include <SFML/Graphics.hpp>
00002 #include <cmath>
00003 //Tile class for tile objects
00004 class Tile {
00005 public:
00006     Tile(const sf::Vector2f& position, const sf::Color& color, int tileSize)
00007     :position(position),color(color),tileSize(tileSize) {
00008         shape.setSize(sf::Vector2f(tileSize, tileSize));
00009         shape.setPosition(position);
00010         shape.setFillColor(color);
00011         if(color == sf::Color::Green){
00012             shape.setOutlineColor(sf::Color::Black);
00013         }
00014         else{
00015             shape.setOutlineColor(color);
00016         }
00017         shape.setOutlineThickness(0.5);
00018     }
00019     //Function to retrieve shape
00020     sf::RectangleShape& getShape() {
00021         return shape;
00022     }
00023     sf::Vector2f& getPosition() {
00024         return position;
00025     }
00026     const sf::Color& getColor() const{
00027         return color;
00028     }
00029
00030 private:
00031     sf::RectangleShape shape;
00032     sf::Vector2f position;
00033     sf::Color color;
00034     int tileSize;
00035 };
00036
00037 Tile& findClosestTile(std::vector<Tile>& tiles, const sf::Vector2f& position) {
00038     Tile* closestTile = nullptr;
00039     float minDistance = std::numeric_limits<float>::max();
00040
00041     for (Tile& tile : tiles) {
00042         sf::Vector2f tileCenter = tile.getPosition()+sf::Vector2f(25,25);
00043         float distance = std::hypot(position.x - tileCenter.x, position.y - tileCenter.y);
00044
00045         if (distance < minDistance) {
00046             minDistance = distance;
00047             closestTile = const_cast<Tile*>(&tile);
00048         }
00049     }
00050
00051     return *closestTile;
00052 }
```

Index