

Tower Defence

Generated by Doxygen 1.9.8

1 Tower Defence	1
2 Hierarchical Index	7
2.1 Class Hierarchy	7
3 Class Index	9
3.1 Class List	9
4 File Index	11
4.1 File List	11
5 Class Documentation	13
5.1 Enemy Class Reference	13
5.1.1 Detailed Description	13
5.1.2 Constructor & Destructor Documentation	14
5.1.2.1 Enemy()	14
5.1.2.2 ~Enemy()	14
5.1.3 Member Function Documentation	14
5.1.3.1 addX()	14
5.1.3.2 addY()	14
5.1.3.3 getHealth()	14
5.1.3.4 getHit()	15
5.1.3.5 getPoints()	15
5.1.3.6 getPosition()	15
5.1.3.7 getRoute()	15
5.1.3.8 getShape()	15
5.1.3.9 getSpeed()	15
5.1.3.10 getXcoord()	15
5.1.3.11 getYcoord()	15
5.1.3.12 hasPassed()	15
5.1.3.13 isDead()	15
5.1.3.14 lowerHealth()	15
5.1.3.15 move()	16
5.1.3.16 moveEnemy()	16
5.1.3.17 reduceSpeed()	16
5.2 EnemyType Class Reference	16
5.2.1 Detailed Description	16
5.2.2 Constructor & Destructor Documentation	17
5.2.2.1 ~EnemyType()	17
5.2.3 Member Function Documentation	17
5.2.3.1 createEnemy()	17
5.3 EnemyTypeA Class Reference	17
5.3.1 Member Function Documentation	18
5.3.1.1 createEnemy()	18

5.4 EnemyTypeB Class Reference	18
5.4.1 Detailed Description	19
5.4.2 Member Function Documentation	19
5.4.2.1 createEnemy()	19
5.5 EnemyTypeC Class Reference	19
5.5.1 Detailed Description	20
5.5.2 Member Function Documentation	20
5.5.2.1 createEnemy()	20
5.6 EnemyTypeD Class Reference	20
5.6.1 Detailed Description	21
5.6.2 Member Function Documentation	21
5.6.2.1 createEnemy()	21
5.7 Tile Class Reference	21
5.7.1 Detailed Description	22
5.7.2 Constructor & Destructor Documentation	22
5.7.2.1 Tile()	22
5.7.3 Member Function Documentation	22
5.7.3.1 getColor()	22
5.7.3.2 getPosition()	23
5.7.3.3 getShape()	23
5.8 Tower Class Reference	23
5.8.1 Constructor & Destructor Documentation	23
5.8.1.1 Tower()	23
5.8.1.2 ~Tower()	24
5.8.2 Member Function Documentation	24
5.8.2.1 addClock()	24
5.8.2.2 attackEnemy()	24
5.8.2.3 getAttack_range()	24
5.8.2.4 getAttackShape()	24
5.8.2.5 getPosition()	24
5.8.2.6 getShape()	24
5.8.2.7 getType()	24
5.9 TowerType Class Reference	25
5.9.1 Detailed Description	25
5.9.2 Constructor & Destructor Documentation	25
5.9.2.1 TowerType()	25
5.9.2.2 ~TowerType()	26
5.9.3 Member Function Documentation	26
5.9.3.1 getAttackRange()	26
5.9.3.2 getAttackSpeed()	26
5.9.3.3 getColor()	26
5.9.3.4 getCost()	26

5.9.3.5 getDamage()	27
5.9.3.6 getRadius()	27
5.10 UniversalClock Class Reference	27
5.10.1 Detailed Description	27
5.10.2 Constructor & Destructor Documentation	28
5.10.2.1 UniversalClock()	28
5.10.3 Member Function Documentation	28
5.10.3.1 isDelayFinished()	28
5.10.3.2 restartClock()	28
6 File Documentation	29
6.1 README.md File Reference	29
6.2 src/gameEngine.hpp File Reference	29
6.3 gameEngine.hpp	29
6.4 src/Graphics/graphicFunctions.cpp File Reference	30
6.4.1 Function Documentation	31
6.4.1.1 addEnemy()	31
6.4.1.2 addTower()	31
6.4.1.3 createButton()	31
6.4.1.4 createText()	32
6.4.1.5 deleteTower()	32
6.4.1.6 drawMoney()	32
6.4.1.7 drawTiles()	33
6.4.1.8 drawTowers()	33
6.4.1.9 drawWave()	33
6.4.1.10 endScreen()	34
6.4.1.11 mainMenu()	34
6.4.1.12 onlyDrawTowers()	34
6.4.1.13 placeTower()	34
6.4.1.14 tutorial()	34
6.4.2 Variable Documentation	35
6.4.2.1 enemies	35
6.4.2.2 exitButton	35
6.4.2.3 playButton	35
6.4.2.4 selectedTowerType	35
6.4.2.5 tiles	35
6.4.2.6 toMain	35
6.4.2.7 towerPlacementMode	35
6.4.2.8 towers	35
6.5 graphicFunctions.cpp	36
6.6 src/main.cpp File Reference	41
6.6.1 Enumeration Type Documentation	41

6.6.1.1 GameState	41
6.6.2 Function Documentation	42
6.6.2.1 main()	42
6.6.2.2 moveEnemies()	42
6.6.3 Variable Documentation	42
6.6.3.1 clock1	42
6.7 src/Objects/enemies.cpp File Reference	42
6.8 src/Objects/enemies.h File Reference	43
6.9 enemies.h	43
6.10 src/Objects/EnemyType.h File Reference	44
6.11 EnemyType.h	44
6.12 src/Objects/EnemyTypeA.cpp File Reference	44
6.13 src/Objects/EnemyTypeA.h File Reference	44
6.14 EnemyTypeA.h	45
6.15 src/Objects/EnemyTypeB.cpp File Reference	45
6.16 src/Objects/EnemyTypeB.h File Reference	45
6.17 EnemyTypeB.h	45
6.18 src/Objects/EnemyTypeC.cpp File Reference	46
6.19 src/Objects/EnemyTypeC.h File Reference	46
6.20 EnemyTypeC.h	46
6.21 src/Objects/EnemyTypeD.cpp File Reference	46
6.22 src/Objects/EnemyTypeD.h File Reference	46
6.23 EnemyTypeD.h	47
6.24 src/Objects/tower.cpp File Reference	47
6.25 src/Objects/tower.h File Reference	47
6.26 tower.h	47
6.27 src/tiles.cpp File Reference	48
6.27.1 Function Documentation	48
6.27.1.1 findClosestTile()	48
6.28 tiles.cpp	49

Index	51
--------------	-----------

Chapter 1

Tower Defence

ELEC-7151 Object oriented programming software project fall 2023

Akseli Tuominen, Nandu Jagdish, Niilo Siren, Noel Nironen

Table of Contents

- 1. Overview
 - 1.1 What the Software Does
- 2. Software Structure
 - 2.1 Overall Architecture
 - 2.2 Class Relationships Diagrams
 - 2.3 Interfaces to External Libraries
- 3. Instructions for Building and Using the Software
 - 3.1 Compilation Instructions
 - 3.1.1 External Library Requirements
 - 3.1.2 Running the software
 - 3.2 Software Usage Guide
- 4. Testing
 - 4.1 Module Testing
 - 4.2 Testing Methods
 - 4.3 Testing Outcomes
- 5. Work Log
 - 5.1 Division of Work and Responsibilities
 - 5.2 Weekly Work Descriptions

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 \`

`libgll-mesa-dev \`

`libegl-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, configuring the game building and compiling (16h)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Enemy	13
EnemyType	16
EnemyTypeA	17
EnemyTypeB	18
EnemyTypeC	19
EnemyTypeD	20
Tile	21
Tower	23
TowerType	25
UniversalClock	27

Chapter 3

Class Index

3.1 Class List

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

Enemy		
Enemy class	13
EnemyType		
The base class for different types of enemies in the game	16
EnemyTypeA	17
EnemyTypeB		
EnemyTypeB.h	18
EnemyTypeC		
EnemyTypeC.h	19
EnemyTypeD		
EnemyTypeD.h	20
Tile		
Represents a tile in the game	21
Tower	23
TowerType		
Tower class	25
UniversalClock		
Clock used for timing in the game engine	27

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

src/ gameEngine.hpp	29
src/ main.cpp	41
src/ tiles.cpp	48
src/Graphics/ graphicFunctions.cpp	30
src/Objects/ enemies.cpp	42
src/Objects/ enemies.h	43
src/Objects/ EnemyType.h	44
src/Objects/ EnemyTypeA.cpp	44
src/Objects/ EnemyTypeA.h	44
src/Objects/ EnemyTypeB.cpp	45
src/Objects/ EnemyTypeB.h	45
src/Objects/ EnemyTypeC.cpp	46
src/Objects/ EnemyTypeC.h	46
src/Objects/ EnemyTypeD.cpp	46
src/Objects/ EnemyTypeD.h	46
src/Objects/ tower.cpp	47
src/Objects/ tower.h	47

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

<i>position</i>	The initial position of the enemy.
<i>radius</i>	The radius of the enemy.
<i>health</i>	The initial health of the enemy.
<i>speed</i>	The speed of the enemy.
<i>x</i>	The x-coordinate of the enemy's position.
<i>y</i>	The y-coordinate of the enemy's position.
<i>color</i>	The color of the enemy.
<i>points</i>	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

<i>h</i>	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

<i>position</i>	The position of the enemy.
<i>x</i>	The x-coordinate of the position.
<i>y</i>	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

<i>position</i>	The position of the enemy.
<i>x</i>	The x-coordinate of the position.
<i>y</i>	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

<i>position</i>	The position of the enemy.
<i>x</i>	The x-coordinate of the position.
<i>y</i>	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

<i>position</i>	The position of the enemy.
<i>x</i>	The x-coordinate of the position.
<i>y</i>	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

<i>position</i>	The position of the enemy.
<i>x</i>	The x-coordinate of the position.
<i>y</i>	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

<i>position</i>	The position of the tile.
<i>color</i>	The color of the tile.
<i>tileSize</i>	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

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

<i>delayTime</i>	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.](#)

```
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
```

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 windowHeight, const int windowHeight, 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

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

<i>window</i>	The game window to add the tower to.
<i>tile</i>	The tile on which the tower is placed.
<i>type</i>	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

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

<i>event</i>	The SFML event that triggered the tower deletion.
<i>window</i>	The game window.
<i>money</i>	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

<i>window</i>	The SFML render window to draw on.
<i>money</i>	The current amount of money.

6.4.1.7 drawTiles()

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

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

Parameters

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

<i>window</i>	The SFML RenderWindow to draw the towers on.
<i>money</i>	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

<i>window</i>	The SFML RenderWindow object to draw on.
<i>gameLevel</i>	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

<i>window</i>	The SFML RenderWindow object used for rendering.
<i>difficulty</i>	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

<i>event</i>	The event that triggered the tower placement.
<i>window</i>	The SFML render window.
<i>money</i>	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.](#)

```

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
00034 void drawTiles(sf::RenderWindow &window, const int tileSize, const int windowHeight, const int
    windowHeight, int difficulty) {
00035     //Count how many tiles we can fit in map
00036     const int mapWidth = windowHeight / tileSize;
00037     const int mapHeight = windowHeight / tileSize;
00038
00039     // Define a hardcoded map, 0 for water(blue), 1 for grass(green) and other for path(white)
00040     int map[16][12];
00041     std::ifstream mapFile;
00042     if(difficulty > 3){
00043         mapFile.open("src/assets/map2.txt");
00044     }
00045     else{
00046         mapFile.open("src/assets/map1.txt");
00047     }
00048
00049     if (!mapFile) {
00050         std::cerr << "Unable to open map file";
00051         // handle error
00052     }
00053
00054     for (int i = 0; i < 16; ++i) {
00055         for (int j = 0; j < 12; ++j) {
00056             char ch;
00057             if (!(mapFile >> ch)) {
00058                 //std::cerr << "Error reading map file";
00059                 // handle error
00060             }
00061             map[i][j] = ch - '0'; // convert char to int
00062         }
00063     }
00064
00065     mapFile.close();
00066
00067     //iterate through all the tiles
00068     for (int x = 0; x < mapWidth; x++) {
00069         for (int y = 0; y < mapHeight; y++) {
00070             //calculate position
00071             sf::Vector2f tilePosition(x * tileSize, y * tileSize);
00072
00073             // Create a Tile object with the specified color, and tileSize
00074             int tileType = map[x][y];
00075             sf::Color tileColor;
00076             if (tileType == 0) {
00077                 tileColor = sf::Color::Blue;
00078             } else if (tileType == 1) {
00079                 tileColor = sf::Color::Green;
00080             } else {
00081                 tileColor = sf::Color::White;
00082             }
00083
00084             //create tile objects for each tile
00085             Tile tile(tilePosition, tileColor, tileSize);
00086
00087             tiles.push_back(tile);
00088             // Draw the tile's shape
00089             window.draw(tile.getShape());
00090         }
    }

```

```

00091     }
00092 }
00093 sf::RectangleShape createButton(float x, float y, float width, float height, sf::Color color) {
00094     sf::RectangleShape button(sf::Vector2f(width, height));
00095     button.setPosition(x, y);
00096     button.setFillColor(color);
00097     return button;
00098 }
00099
00100 // Function to create a text
00112 sf::Text createText(float x, float y, std::string content, sf::Font& font, unsigned int size, sf::Color
color) {
00113     sf::Text text;
00114     text.setFont(font);
00115     text.setFillColor(color);
00116     text.setString(content);
00117     text.setCharacterSize(size);
00118     text.setPosition(x, y);
00119     return text;
00120 }
00121
00129 void addTower(sf::RenderWindow &window, Tile tile, TowerType type){
00130     Tower tower(tile.getPosition(), type);
00131     towers.push_back(tower);
00132 }
00133
00144 void addEnemy(sf::RenderWindow &window, int tileSize, int x, int y, int gameLevel, int difficulty){
00145     int iterator = gameLevel * difficulty;
00146
00147     if (gameLevel <= 2) {
00148         for (int j = 1; j < 3*iterator; j++) {
00149
00150             sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
00151             EnemyTypeA enemyTypeA;
00152             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
00153         }
00154     }
00155     else if (gameLevel > 2 && gameLevel <= 4) {
00156         for (int j = 1; j < 3*iterator; j++) {
00157             sf::Vector2f tileStartPosition_B((-j-5)*tileSize+7, y * tileSize+7);
00158             EnemyTypeB enemyTypeB;
00159             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
(-j-5)*tileSize,y*tileSize));
00160         }
00161         for (int j = 1; j < 3*iterator; j++) {
00162             sf::Vector2f tileStartPosition_A((-j)*tileSize+4, y * tileSize+4);
00163             EnemyTypeA enemyTypeA;
00164             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A, (-j)*tileSize,y*tileSize));
00165         }
00166     }
00167     else if (gameLevel > 4 && gameLevel <= 6) {
00168         sf::Vector2f tileStartPosition_D((x-10.5)*tileSize+1, y * tileSize+1);
00169         EnemyTypeD enemyTypeD;
00170         enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-10.5)*tileSize,y*tileSize));
00171
00172         for (int j = 1; j < 4*iterator; j++) {
00173             sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
00174             EnemyTypeB enemyTypeB;
00175             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
(-j-15)*tileSize,y*tileSize));
00176         }
00177         for (int j = 1; j < 2*iterator; j++){
00178             sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
00179             EnemyTypeA enemyTypeA;
00180             enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A,
(-j-6)*tileSize,y*tileSize));
00181         }
00182         for (int j = 1; j < 1*iterator; j++){
00183             sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
00184             EnemyTypeC enemyTypeC;
00185             enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C,
(-j-2)*tileSize,y*tileSize));
00186         }
00187     }
00188     else if (gameLevel > 6) {
00189         sf::Vector2f tileStartPosition_D((x-11.5)*tileSize+1, y * tileSize+1);
00190         EnemyTypeD enemyTypeD;
00191         enemies.push_back(enemyTypeD.createEnemy(tileStartPosition_D, (x-11.5)*tileSize,y*tileSize));
00192
00193         for (int j = 1; j < 8*iterator; j++) {
00194             sf::Vector2f tileStartPosition_B((-j-15)*tileSize+7, y * tileSize+7);
00195             EnemyTypeB enemyTypeB;
00196             enemies.push_back(enemyTypeB.createEnemy(tileStartPosition_B,
(-j-15)*tileSize,y*tileSize));
00197         }
00198         for (int j = 1; j < 4*iterator; j++){
00199

```

```

00200         sf::Vector2f tileStartPosition_A((-j-6)*tileSize+4, y * tileSize+4);
00201         EnemyTypeA enemyTypeA;
00202         enemies.push_back(enemyTypeA.createEnemy(tileStartPosition_A,
(-j-6)*tileSize,y*tileSize));
00203     }
00204     for (int j = 1; j < 2*iterator; j++){
00205         sf::Vector2f tileStartPosition_C((-j-2) * tileSize+1, y * tileSize+1);
00206         EnemyTypeC enemyTypeC;
00207         enemies.push_back(enemyTypeC.createEnemy(tileStartPosition_C,
(-j-2)*tileSize,y*tileSize));
00208     }
00209 }
00210
00211 }
00212 }
00213
00214
00215
00223 void placeTower(sf::Event event, sf::RenderWindow &window, int &money){
00224
00225     sf::Font font;
00226     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00227         std::cout << "Could not load font" << std::endl;
00228     }
00229
00230     //Define the tower types
00231     TowerType basicTower(30.0, 20, 100, 50.0, sf::Color::Red,30);
00232     TowerType advancedTower(40.0, 30, 100, 60.0, sf::Color::Blue,50);
00233     TowerType ultimateTower(50.0, 40, 250, 70.0, sf::Color::Yellow,120);
00234     TowerType cashTower(0,0,100,50,sf::Color::Black,300);
00235
00236     // Create the buttons for each tower type
00237     sf::RectangleShape basicButton = createButton(50, 500, 50, 50, basicTower.getColor());
00238     sf::RectangleShape advancedButton = createButton(0, 500, 50, 50, advancedTower.getColor());
00239     sf::RectangleShape ultimateButton = createButton(50, 550, 50, 50, ultimateTower.getColor());
00240     sf::RectangleShape cashButton = createButton(100, 550, 50, 50, cashTower.getColor());
00241
00242     sf::Text cost1 = createText(52, 502, std::to_string(30) + "$", font, 20, sf::Color::Black);
00243     sf::Text cost2 = createText(2, 502, std::to_string(50) + "$", font, 20, sf::Color::Black);
00244     sf::Text cost3 = createText(52, 552, std::to_string(120) + "$", font, 20, sf::Color::Black);
00245     sf::Text cost4 = createText(102, 552, std::to_string(300) + "$", font, 20, sf::Color::White);
00246
00247     // Draw the buttons
00248     window.draw(basicButton);
00249     window.draw(advancedButton);
00250     window.draw(ultimateButton);
00251     window.draw(cashButton);
00252
00253     // Draw the cost text
00254     window.draw(cost1);
00255     window.draw(cost2);
00256     window.draw(cost3);
00257     window.draw(cost4);
00258
00259
00260     // Check if a button was clicked
00261     if (event.type == sf::Event::MouseButtonPressed &&
event.mouseButton.button == sf::Mouse::Left) {
00262         if (basicButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
00263             selectedTowerType = &basicTower;
00264         } else if (advancedButton.getGlobalBounds().contains(event.mouseButton.x,
event.mouseButton.y)) {
00265             selectedTowerType = &advancedTower;
00266         } else if (ultimateButton.getGlobalBounds().contains(event.mouseButton.x,
event.mouseButton.y)) {
00267             selectedTowerType = &ultimateTower;
00268         } else if (cashButton.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y)) {
00269             selectedTowerType = &cashTower;
00270         }
00271     }
00272 }
00273
00274 // Check if a tile was clicked
00275 if (selectedTowerType != nullptr && event.mouseButton.button == sf::Mouse::Right) {
00276     sf::Vector2i mousePos = sf::Mouse::getPosition(window);
00277     sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
00278     Tile& closestTile = findClosestTile(tiles, rightPosition);
00279
00280     if (closestTile.getColor() == sf::Color::Green) {
00281         if (money < selectedTowerType->getCost()) {
00282             std::cout << "Not enough money" << std::endl;
00283         }
00284         else{
00285             money -= selectedTowerType->getCost();
00286         }
00287     }
00288 }
00289

```

```

00290
00291         addTower(window, closestTile, *selectedTowerType);
00292
00293
00294         selectedTowerType = nullptr;
00295     }
00296 }
00297 }
00298 }
00299
00300 void drawTowers(sf::RenderWindow &window, int &money){
00301     std::random_device rd;
00302     std::mt19937 gen(rd());
00303     std::uniform_int_distribution<> distrib(0, 20);
00304
00305     for (int i=0; i<towers.size();i++) {
00306         window.draw(towers[i].getShape());
00307         if (towers[i].attackEnemy(enemies)==1) {
00308             towers.erase(towers.begin() + i);
00309             std::cout << "Removing successful" << std::endl;
00310         }
00311         if (towers[i].attackEnemy(enemies)==2 && distrib(gen) == 1) {
00312             money += 1;
00313         }
00314     }
00315 }
00316
00317 void onlyDrawTowers(sf::RenderWindow &window){
00318     for (int i=0; i<towers.size();i++){
00319         window.draw(towers[i].getShape());
00320         window.draw(towers[i].getAttackShape());
00321     }
00322 }
00323
00324 void mainMenu(sf::RenderWindow &window, int difficulty) {
00325     window.clear();
00326     sf::Font font;
00327     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00328         std::cout << "Could not load font" << std::endl;
00329     }
00330
00331     std::vector<std::string> options = {"Easy", "Medium", "Hard", "Very Hard", "Insane"};
00332     std::vector<sf::RectangleShape> buttons;
00333     std::vector<sf::Text> texts;
00334
00335     for (int i = 0; i < options.size(); i++) {
00336         buttons.push_back(createButton(300, 100 + i * 60, 200, 50, sf::Color::Blue));
00337         texts.push_back(createText(350, 110 + i * 60, options[i], font, 24, sf::Color::White));
00338     }
00339     //Create Tutorial button and text
00340     buttons.push_back(createButton(300, 40, 200, 50, sf::Color::Green));
00341     texts.push_back(createText(350, 50, "Tutorial", font, 24, sf::Color::Black));
00342
00343     //Create Play button and text
00344     buttons.push_back(createButton(300, 400, 200, 50, sf::Color::Green));
00345     texts.push_back(createText(350, 410, "Play", font, 24, sf::Color::Black));
00346
00347     //Create Exit button and text
00348     buttons.push_back(createButton(300, 460, 200, 50, sf::Color::Red));
00349     texts.push_back(createText(350, 470, "Exit", font, 24, sf::Color::White));
00350
00351     for (int i = 0; i < buttons.size(); i++) {
00352         if(difficulty == i+1){
00353             buttons[i].setFillColor(sf::Color::Yellow);
00354             texts[i].setFillColor(sf::Color::Black);
00355         }
00356         window.draw(buttons[i]);
00357         window.draw(texts[i]);
00358     }
00359 }
00360
00361 void drawMoney(sf::RenderWindow &window, int money) {
00362     sf::Font font;
00363     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00364         std::cout << "Could not load font" << std::endl;
00365     }
00366
00367     std::string moneyString = std::to_string(money) + "$";
00368     sf::Text moneyText = createText(680, -10, moneyString, font, 50, sf::Color::Yellow);
00369
00370     if (money > 99 && money < 1000) {

```

```

00395         sf::Text moneyText = createText(660, -10, moneyString, font, 50, sf::Color::Yellow);
00396     }
00397     else if (money >= 1000) {
00398         sf::Text moneyText = createText(635, -10, moneyString, font, 50, sf::Color::Yellow);
00399     }
00400
00401     window.draw(moneyText);
00402 }
00403 void endScreen(sf::RenderWindow &window) {
00404     sf::Font font;
00405     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00406         std::cout << "Could not load font" << std::endl;
00407     }
00408
00409     sf::Text text = createText(250, 200, "You lost!", font, 50, sf::Color::Red);
00410     sf::Text text2 = createText(20, 300, "Try with easier difficulty ;)", font, 45, sf::Color::Red);
00411     sf::Text text3 = createText(150, 400, "Click to try again", font, 45, sf::Color::Red);
00412     window.clear();
00413     window.draw(text);
00414     window.draw(text2);
00415     window.draw(text3);
00416 }
00427 void deleteTower(sf::Event event, sf::RenderWindow &window, int &money){
00428     if (event.type == sf::Event::MouseButtonPressed &&
00429         event.mouseButton.button == sf::Mouse::Left) {
00430         sf::Vector2i mousePos = sf::Mouse::getPosition(window);
00431         sf::Vector2f rightPosition(static_cast<float>(mousePos.x), static_cast<float>(mousePos.y));
00432         Tile& closestTile = findClosestTile(tiles, rightPosition);
00433         for (int i = 0; i < towers.size(); i++) {
00434             if (towers[i].getPosition() == closestTile.getPosition()) {
00435                 money += towers[i].getType().getCost()/2;
00436                 towers.erase(towers.begin() + i);
00437             }
00438         }
00439     }
00440 }
00441
00442 void tutorial(sf::RenderWindow &window){
00443     window.clear();
00444     sf::Text tutorialText;
00445     sf::Font font;
00446     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00447         std::cout << "Could not load font" << std::endl;
00448     }
00449     tutorialText.setFont(font);
00450     tutorialText.setString("Welcome to the game! Here's how to play:\n\n"
00451         "1. Start by selecting a difficulty level.\n"
00452         "2. Press the 'Play' button to start the game.\n"
00453         "3. The game consists of two phases: Building and Attacking.\n"
00454         "4. During the Building phase, you can build or remove towers.\n"
00455         "5. To build a tower, first left-click a colored square to select the type of\n"
00456         "tower. \n Then, right-click on a green tile to place the tower. \n Make sure you have enough money\n"
00457         "to buy the tower!\n\n"
00458         "6. You can remove a tower by left-clicking it during the Building phase. \n"
00459         "You'll get some money back when you do this.\n Range of each tower is visible in building stage.\n"
00460         "7. When you're ready, press the black cube to switch to the Attacking phase.\n"
00461         "Once no enemies are left standing\n building phase starts again, Good luck! \n\n"
00462         "Towers\n"
00463         "1. Red Tower: Cost: 30$, Damage: 20, Range: 100, Attack Speed: 50,\n special\n"
00464         "skill: none\n"
00465         "2. Blue Tower: Cost: 50$, Damage: 30, Range: 100, Attack Speed: 60,\n"
00466         "special skill: slows enemies\n"
00467         "3. Yellow Tower: Cost: 120$, Damage: 40, Range: 250, Attack Speed: 70,\n"
00468         "special skill: Huge range and Damage \n"
00469         "4. Black Tower: Cost: 300$, Damage: 0, Range: 100, Attack Speed: 50,\n"
00470         "special skill: Gives you money\n\n"
00471         "Enemies\n"
00472         "1. Red Enemy: Health: 300, Speed: 2.5, Reward: 1\n special skill: none\n"
00473         "2. Cyan Enemy: Health: 100, Speed: 5, Reward: 2\n special skill: very\n"
00474         "fast\n"
00475         "3. Black Enemy: Health: 2500, Speed: 1, Reward: 5\n special skill: very\n"
00476         "tanky\n"
00477         "4. Yellow Enemy: Health: 2500, Speed: 1, Reward: 0\n special skill: can\n"
00478         "destroy red towers\n");
00479     tutorialText.setCharacterSize(15);
00480     tutorialText.setFill(sf::Color::White);
00481     tutorialText.setPosition(50, 5);
00482
00483     sf::RectangleShape backButton = createButton(100, 540, 200, 50, sf::Color::Blue);
00484     sf::Text text = createText(150, 545, "Back", font, 30, sf::Color::White);
00485
00486     window.draw(backButton);
00487     window.draw(text);
00488     window.draw(tutorialText);
00489     if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
00490         sf::Vector2i mousePos = sf::Mouse::getPosition(window);

```

```

00481         if (backButton.getGlobalBounds().contains(mousePos.x, mousePos.y)) {
00482             toMain = true;
00483         }
00484     }
00485 }
00486
00487
00488
00495 void drawWave(sf::RenderWindow &window, int &gameLevel) {
00496     sf::Font font;
00497     if (!font.loadFromFile("src/assets/FreeMono.ttf")) {
00498         std::cout << "Could not load font" << std::endl;
00499     }
00500
00501     std::string waveString = "Wave: " + std::to_string(gameLevel-1);
00502
00503     sf::Text text = createText(290, -10, waveString, font, 50, sf::Color::Black);
00504     window.draw(text);
00505 }

```

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

<i>clock</i>	The UniversalClock object used to track the delay time.
<i>window</i>	The <code>sf::RenderWindow</code> object used to draw the enemies.
<i>stored_enemies</i>	The vector of Enemy objects representing the enemies on the screen.
<i>delayTime</i>	The delay time in seconds between enemy movements.
<i>Money</i>	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.](#)

```
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
00022 class Enemy {
00023 public:
00035     Enemy(sf::Vector2f& position, double radius, int health, double speed, float x, float y,
sf::Color& color,int points);
00036     sf::CircleShape& getShape();
00037     sf::Vector2f& getPosition();
00038
00039
00040     void move(float x_dir, float y_dir);
00041     void moveEnemy(double timeStep, sf::RenderWindow &window);
00042
00043     int getRoute();
00044     int getSpeed();
00045     int getXcoord();
00046     int getYcoord();
00047
00048     void addY(int b);
00049     void addX(int a);
00050
00056     void lowerHealth(int h);
00057
00058     bool hasPassed();
00059     bool isDead();
00060
00061     void getHit(int damage){
00062         health -= damage;
00063     }
00064
00065     int getHealth(){
00066         return this->health;
00067     }
00068     int getPoints(){
00069         return this->points;
00070     }
00076     void reduceSpeed(){
00077         if(this->shape.getFillColor() == sf::Color::Cyan){
00078             speed = 2;
00079         }
00080         else if(this->shape.getFillColor() == sf::Color::Black){
00081             speed = 1;
```

```

00082         }
00083         else if(this->shape.getFillColor() == sf::Color::Red){
00084             speed = 1;
00085         }
00086     }
00087     // destructor
00088     ~Enemy(){
00089     }
00090 }
00091 private:
00092     sf::CircleShape shape;
00093     sf::Vector2f position;
00094     int x;
00095     int y;
00096     float speed;
00097     int health;
00098     int points;
00099     int route;
00100 };
00101
00102 #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.](#)

```

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

```

6.12 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.](#)

```
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
```

6.15 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.](#)

```
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
```

6.18 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.](#)

```
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
```

6.21 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.](#)

```
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
```

6.24 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.](#)

```
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
00010
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

```

6.27 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

<i>tiles</i>	The vector of Tiles to search from.
<i>position</i>	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.](#)

```

00001 #include <SFML/Graphics.hpp>
00002 #include <cmath>
00003 //Tile class for tile objects
00007 class Tile {
00008 public:
00015     Tile(const sf::Vector2f& position, const sf::Color& color, int tileSize)
00016     :position(position),color(color),tileSize(tileSize) {
00017         shape.setSize(sf::Vector2f(tileSize, tileSize));
00018         shape.setPosition(position);
00019         shape.setFillColor(color);
00020         if(color == sf::Color::Green){
00021             shape.setOutlineColor(sf::Color::Black);
00022         }
00023         else{
00024             shape.setOutlineColor(color);
00025         }
00026         shape.setOutlineThickness(0.5);
00027     }
00028
00033     sf::RectangleShape& getShape() {
00034         return shape;
00035     }
00036
00041     sf::Vector2f& getPosition() {
00042         return position;
00043     }
00044
00049     const sf::Color& getColor() const{
00050         return color;
00051     }
00052
00053 private:
00054     sf::RectangleShape shape;
00055     sf::Vector2f position;
00056     sf::Color color;
00057     int tileSize;
00058 };
00059
00067 Tile& findClosestTile(std::vector<Tile>& tiles, const sf::Vector2f& position) {
00068     Tile* closestTile = nullptr;
00069     float minDistance = std::numeric_limits<float>::max();
00070
00071     for (Tile& tile : tiles) {
00072         sf::Vector2f tileCenter = tile.getPosition()+sf::Vector2f(25,25);
00073         float distance = std::hypot(position.x - tileCenter.x, position.y - tileCenter.y);
00074
00075         if (distance < minDistance) {
00076             minDistance = distance;
00077             closestTile = const_cast<Tile*>(&tile);
00078         }
00079     }
00080
00081     return *closestTile;
00082 }

```


Index

- ~Enemy
 - Enemy, [14](#)
- ~EnemyType
 - EnemyType, [17](#)
- ~Tower
 - Tower, [23](#)
- ~TowerType
 - TowerType, [26](#)
- addClock
 - Tower, [24](#)
- addEnemy
 - graphicFunctions.cpp, [31](#)
- addTower
 - graphicFunctions.cpp, [31](#)
- addX
 - Enemy, [14](#)
- addY
 - Enemy, [14](#)
- attackEnemy
 - Tower, [24](#)
- Attacking
 - main.cpp, [42](#)
- Building
 - main.cpp, [42](#)
- clock1
 - main.cpp, [42](#)
- createButton
 - graphicFunctions.cpp, [31](#)
- createEnemy
 - EnemyType, [17](#)
 - EnemyTypeA, [18](#)
 - EnemyTypeB, [19](#)
 - EnemyTypeC, [20](#)
 - EnemyTypeD, [21](#)
- createText
 - graphicFunctions.cpp, [31](#)
- deleteTower
 - graphicFunctions.cpp, [32](#)
- drawMoney
 - graphicFunctions.cpp, [32](#)
- drawTiles
 - graphicFunctions.cpp, [33](#)
- drawTowers
 - graphicFunctions.cpp, [33](#)
- drawWave
 - graphicFunctions.cpp, [33](#)
- EndScreen
 - main.cpp, [42](#)
- endScreen
 - graphicFunctions.cpp, [34](#)
- enemies
 - graphicFunctions.cpp, [35](#)
- Enemy, [13](#)
 - ~Enemy, [14](#)
 - addX, [14](#)
 - addY, [14](#)
 - Enemy, [14](#)
 - getHealth, [14](#)
 - getHit, [14](#)
 - getPoints, [15](#)
 - getPosition, [15](#)
 - getRoute, [15](#)
 - getShape, [15](#)
 - getSpeed, [15](#)
 - getXcoord, [15](#)
 - getYcoord, [15](#)
 - hasPassed, [15](#)
 - isDead, [15](#)
 - lowerHealth, [15](#)
 - move, [16](#)
 - moveEnemy, [16](#)
 - reduceSpeed, [16](#)
- EnemyType, [16](#)
 - ~EnemyType, [17](#)
 - createEnemy, [17](#)
- EnemyTypeA, [17](#)
 - createEnemy, [18](#)
- EnemyTypeB, [18](#)
 - createEnemy, [19](#)
- EnemyTypeC, [19](#)
 - createEnemy, [20](#)
- EnemyTypeD, [20](#)
 - createEnemy, [21](#)
- exitButton
 - graphicFunctions.cpp, [35](#)
- findClosestTile
 - tiles.cpp, [48](#)
- GameState
 - main.cpp, [41](#)
- getAttack_range
 - Tower, [24](#)
- getAttackRange
 - TowerType, [26](#)
- getAttackShape

- Tower, [24](#)
- getAttackSpeed
 - TowerType, [26](#)
- getColor
 - Tile, [22](#)
 - TowerType, [26](#)
- getCost
 - TowerType, [26](#)
- getDamage
 - TowerType, [26](#)
- getHealth
 - Enemy, [14](#)
- getHit
 - Enemy, [14](#)
- getPoints
 - Enemy, [15](#)
- getPosition
 - Enemy, [15](#)
 - Tile, [22](#)
 - Tower, [24](#)
- getRadius
 - TowerType, [27](#)
- getRoute
 - Enemy, [15](#)
- getShape
 - Enemy, [15](#)
 - Tile, [23](#)
 - Tower, [24](#)
- getSpeed
 - Enemy, [15](#)
- getType
 - Tower, [24](#)
- getXcoord
 - Enemy, [15](#)
- getYcoord
 - Enemy, [15](#)
- graphicFunctions.cpp
 - addEnemy, [31](#)
 - addTower, [31](#)
 - createButton, [31](#)
 - createText, [31](#)
 - deleteTower, [32](#)
 - drawMoney, [32](#)
 - drawTiles, [33](#)
 - drawTowers, [33](#)
 - drawWave, [33](#)
 - endScreen, [34](#)
 - enemies, [35](#)
 - exitButton, [35](#)
 - mainMenu, [34](#)
 - onlyDrawTowers, [34](#)
 - placeTower, [34](#)
 - playButton, [35](#)
 - selectedTowerType, [35](#)
 - tiles, [35](#)
 - toMain, [35](#)
 - towerPlacementMode, [35](#)
 - towers, [35](#)
 - tutorial, [34](#)
- hasPassed
 - Enemy, [15](#)
- isDead
 - Enemy, [15](#)
- isDelayFinished
 - UniversalClock, [28](#)
- lowerHealth
 - Enemy, [15](#)
- main
 - main.cpp, [42](#)
- main.cpp
 - Attacking, [42](#)
 - Building, [42](#)
 - clock1, [42](#)
 - EndScreen, [42](#)
 - GameState, [41](#)
 - main, [42](#)
 - MainMenu, [42](#)
 - moveEnemies, [42](#)
 - Tutorial, [42](#)
- MainMenu
 - main.cpp, [42](#)
- mainMenu
 - graphicFunctions.cpp, [34](#)
- move
 - Enemy, [16](#)
- moveEnemies
 - main.cpp, [42](#)
- moveEnemy
 - Enemy, [16](#)
- onlyDrawTowers
 - graphicFunctions.cpp, [34](#)
- placeTower
 - graphicFunctions.cpp, [34](#)
- playButton
 - graphicFunctions.cpp, [35](#)
- README.md, [29](#)
- reduceSpeed
 - Enemy, [16](#)
- restartClock
 - UniversalClock, [28](#)
- selectedTowerType
 - graphicFunctions.cpp, [35](#)
- src/gameEngine.hpp, [29](#)
- src/Graphics/graphicFunctions.cpp, [30](#), [36](#)
- src/main.cpp, [41](#)
- src/Objects/enemies.cpp, [42](#)
- src/Objects/enemies.h, [43](#)
- src/Objects/EnemyType.h, [44](#)
- src/Objects/EnemyTypeA.cpp, [44](#)
- src/Objects/EnemyTypeA.h, [44](#), [45](#)

- src/Objects/EnemyTypeB.cpp, [45](#)
- src/Objects/EnemyTypeB.h, [45](#)
- src/Objects/EnemyTypeC.cpp, [46](#)
- src/Objects/EnemyTypeC.h, [46](#)
- src/Objects/EnemyTypeD.cpp, [46](#)
- src/Objects/EnemyTypeD.h, [46](#), [47](#)
- src/Objects/tower.cpp, [47](#)
- src/Objects/tower.h, [47](#)
- src/tiles.cpp, [48](#), [49](#)

- Tile, [21](#)
 - getColor, [22](#)
 - getPosition, [22](#)
 - getShape, [23](#)
 - Tile, [22](#)
- tiles
 - graphicFunctions.cpp, [35](#)
- tiles.cpp
 - findClosestTile, [48](#)
- toMain
 - graphicFunctions.cpp, [35](#)
- Tower, [23](#)
 - ~Tower, [23](#)
 - addClock, [24](#)
 - attackEnemy, [24](#)
 - getAttack_range, [24](#)
 - getAttackShape, [24](#)
 - getPosition, [24](#)
 - getShape, [24](#)
 - getType, [24](#)
 - Tower, [23](#)
- Tower Defence, [1](#)
- towerPlacementMode
 - graphicFunctions.cpp, [35](#)
- towers
 - graphicFunctions.cpp, [35](#)
- TowerType, [25](#)
 - ~TowerType, [26](#)
 - getAttackRange, [26](#)
 - getAttackSpeed, [26](#)
 - getColor, [26](#)
 - getCost, [26](#)
 - getDamage, [26](#)
 - getRadius, [27](#)
 - TowerType, [25](#)
- Tutorial
 - main.cpp, [42](#)
- tutorial
 - graphicFunctions.cpp, [34](#)

- UniversalClock, [27](#)
 - isDelayFinished, [28](#)
 - restartClock, [28](#)
 - UniversalClock, [28](#)