Tower Defense

Generated by Doxygen 1.9.8

Chapter 1

Source content

This folder should contain only hpp/cpp files of your implementation. You can also place hpp files in a separate directory `include`.

You can create a summary of files here. It might be useful to describe file relations, and brief summary of their content.

Chapter 2

Test files

It is a common practice to do unit tests of each class before you integrate it into the project to validate its operation. In this folder, you can create your own unit test files to validate the operation of your components.

> It might be a good idea to also take some notes about the tests since you are required to report these in the final report.

2.1 Unit Tests

2.1.1 Test of MyClass

Involved Classes:

Test File:

Results:

Chapter 3

Namespace Index

3.1 Namespace List

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

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

Namespace Documentation

7.1 GameState Namespace Reference

enum for game states.

Enumerations

- enum **State** {
  StartMenu , **MapMenu** , **Pause** , **Round** ,
  Victory , **GameOver** }

7.1.1 Detailed Description

enum for game states.

7.2 ObjectTypes Namespace Reference

Namespace containing enums for different enemy and tower types.

Enumerations

- enum Enemies {
  NoobSkeleton_NoAttack , **NoobDemon_CanAttack** , **FastBoy** , **FogMage** ,
  HealerPriest , **InfernoMage** , **TankOrc** , **BossKnight** }

  Enums representing different types of enemies.
- enum Towers {
  AoeTower , **ArcherTower** , **MudMageTower** , **RepelMageTower** ,
  SniperTower , **WaterMageTower** }

  Enums representing different types of towers.

7.2.1 Detailed Description

Namespace containing enums for different enemy and tower types.

7.3 TowerAttributes Namespace Reference

An enumerator for the tower attributes.

Enumerations

- enum **Atr** {
  HP , **DMG** , **RNG** , **ATKSPD** ,
  MONEY , **ROUND** }

7.3.1 Detailed Description

An enumerator for the tower attributes.

Chapter 8

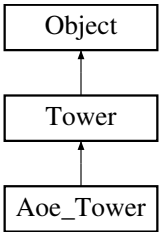
Class Documentation

8.1 Aoe_Tower Class Reference

Class for AOE tower.

`#include <aoe_tower.hpp>`

Inheritance diagram for Aoe_Tower:



Public Member Functions

- Aoe_Tower (Level &current_level, Vector2D &position, int health=400, int damage=50, int range=200, int attack_speed=25, int type=ObjectTypes::AoeTower, int price=150, int level=1)

  Constructor to initialize an AOE Tower object.
- ~**Aoe_Tower** ()

  Destructor for the AOE Tower.
- bool attack ()

  Perform an attack action specific to the AOE Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.1.1 Detailed Description

Class for AOE tower.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 Aoe_Tower()

```
Aoe_Tower::Aoe_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 400,
            int damage = 50,
            int range = 200,
            int attack_speed = 25,
            int type = ObjectTypes::AoeTower,
            int price = 150,
            int level = 1 )
```

Constructor to initialize an AOE Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the AOE Tower (Vector2D).
health	Initial health points of the AOE Tower.
damage	Damage inflicted by the AOE Tower.
range	Range of attack for the AOE Tower.
attack_speed	Speed of attack for the AOE Tower.
type	Type of the AOE Tower.
price	Price of the AOE Tower.
level	Level of the AOE Tower.

8.1.3 Member Function Documentation

8.1.3.1 attack()

```
bool Aoe_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the AOE Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

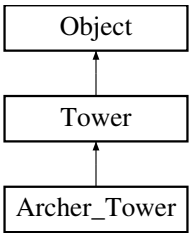
- src/aoe_tower.hpp
- src/aoe_tower.cpp

8.2 Archer_Tower Class Reference

Class for Archer tower.

`#include <archer_tower.hpp>`

Inheritance diagram for Archer_Tower:



Public Member Functions

- Archer_Tower (Level &current_level, Vector2D &position, int health=300, int damage=25, int range=250, int attack_speed=20, int type=ObjectTypes::ArcherTower, int price=100, int level=1)

  Constructor to initialize an Archer Tower object.
- ~**Archer_Tower** ()

  Destructor for the Archer Tower.
- bool attack ()

  Perform an attack action specific to the Archer Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.2.1 Detailed Description

Class for Archer tower.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Archer_Tower()

```
Archer_Tower::Archer_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 300,
            int damage = 25,
            int range = 250,
            int attack_speed = 20,
            int type = ObjectTypes::ArcherTower,
            int price = 100,
            int level = 1 )
```

Constructor to initialize an Archer Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Archer Tower (Vector2D).
health	Initial health points of the Archer Tower.
damage	Damage inflicted by the Archer Tower.
range	Range of attack for the Archer Tower.
attack_speed	Speed of attack for the Archer Tower.
type	Type of the Archer Tower.
price	Price of the Archer Tower.
level	Level of the Archer Tower.

8.2.3 Member Function Documentation

8.2.3.1 attack()

```
bool Archer_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Archer Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

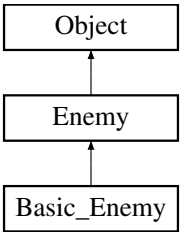
- src/archer_tower.hpp
- src/archer_tower.cpp

8.3 Basic_Enemy Class Reference

Test class for Basic enemy.

`#include <basic_enemy.hpp>`

Inheritance diagram for Basic_Enemy:



Public Member Functions

- Basic_Enemy (Level &level, Vector2D &position, int health=20, int damage=5, int range=100, int attack_↩ speed=1, int type=ObjectTypes::NoobDemon_CanAttack, int speed=20, int defense=5, int size=3)

  Constructor to initialize a Basic Enemy object.
- ~**Basic_Enemy** ()

  Destructor for the Basic Enemy.
- bool attack ()

  Perform an attack action specific to the Basic Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.3.1 Detailed Description

Test class for Basic enemy.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Basic_Enemy()

```
Basic_Enemy::Basic_Enemy (
            Level & level,
            Vector2D & position,
            int health = 20,
            int damage = 5,
            int range = 100,
            int attack_speed = 1,
            int type = ObjectTypes::NoobDemon_CanAttack,
            int speed = 20,
            int defense = 5,
            int size = 3 )
```

Constructor to initialize a Basic Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Basic Enemy (Vector2D).
health	Initial health points of the Basic Enemy.
damage	Damage inflicted by the Basic Enemy.
range	Range of attack for the Basic Enemy.
attack_speed	Speed of attack for the Basic Enemy.
type	Type of the Basic Enemy.
speed	Speed attribute of the Basic Enemy.
defense	Defense attribute of the Basic Enemy.
size	Size of the Basic Enemy.

8.3.3 Member Function Documentation

8.3.3.1 attack()

```
bool Basic_Enemy::attack ( )  [virtual]
```

Perform an attack action specific to the Basic Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

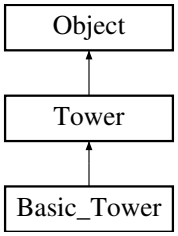
- src/basic_enemy.hpp
- src/basic_enemy.cpp

8.4 Basic_Tower Class Reference

Test class for a Basic Tower.

`#include <basic_tower.hpp>`

Inheritance diagram for Basic_Tower:



Public Member Functions

- Basic_Tower (Level &current_level, Vector2D &position, int health=30, int damage=10, int range=100, int attack_speed=1, int type=ObjectTypes::ArcherTower, int price=100, int level=1, bool attack_type_single=true)

  Constructor to initialize a Basic Tower object.
- ~**Basic_Tower** ()

  Destructor for the Basic Tower.
- bool attack ()

  Perform an attack action specific to the Basic Tower.
- void **set_multiple_target** ()

  Set the attack type to target multiple enemies.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.4.1 Detailed Description

Test class for a Basic Tower.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Basic_Tower()

```
Basic_Tower::Basic_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 30,
            int damage = 10,
            int range = 100,
            int attack_speed = 1,
            int type = ObjectTypes::ArcherTower,
            int price = 100,
            int level = 1,
            bool attack_type_single = true )
```

Constructor to initialize a Basic Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Basic Tower (Vector2D).
health	Initial health points of the Basic Tower.
damage	Damage inflicted by the Basic Tower.
range	Range of attack for the Basic Tower.
attack_speed	Speed of attack for the Basic Tower.
type	Type of the Basic Tower.
price	Price of the Basic Tower.
level	Level of the Basic Tower.
attack_type_single	Flag indicating single or multiple target attack type.

8.4.3 Member Function Documentation

8.4.3.1 attack()

```
bool Basic_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Basic Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

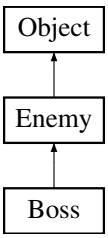
- src/basic_tower.hpp
- src/basic_tower.cpp

8.5 Boss Class Reference

Class for a Boss enemy.

```
#include <boss_enemy.hpp>
```

Inheritance diagram for Boss:



Public Member Functions

- Boss (Level &level, Vector2D &position, int health=800, int damage=20, int range=150, int attack_speed=40, int type=ObjectTypes::BossKnight, int speed=1, int defense=20, int size=5)

  Constructor to initialize a Boss Enemy object.
- ~**Boss** ()

  Destructor for the Boss Enemy.
- bool attack ()

  Perform an attack action specific to the Boss Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.5.1 Detailed Description

Class for a Boss enemy.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Boss()

```
Boss::Boss (
            Level & level,
            Vector2D & position,
            int health = 800,
            int damage = 20,
            int range = 150,
            int attack_speed = 40,
            int type = ObjectTypes::BossKnight,
            int speed = 1,
            int defense = 20,
            int size = 5 )
```

Constructor to initialize a Boss Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Boss Enemy (Vector2D).
health	Initial health points of the Boss Enemy.
damage	Damage inflicted by the Boss Enemy.
range	Range of attack for the Boss Enemy.
attack_speed	Speed of attack for the Boss Enemy.
type	Type of the Boss Enemy.
speed	Speed attribute of the Boss Enemy.
defense	Defense attribute of the Boss Enemy.
size	Size of the Boss Enemy.

8.5.3 Member Function Documentation

8.5.3.1 attack()

```
bool Boss::attack ( )  [virtual]
```

Perform an attack action specific to the Boss Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

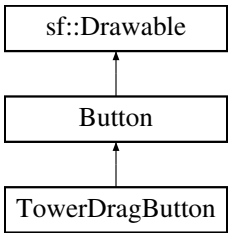
- src/boss_enemy.hpp
- src/boss_enemy.cpp

8.6 Button Class Reference

A class to represent buttons in menus. Inherits from the sfml Drawable class to hide draw calls. Works as a transparent button in the two first menus and as an opaque button for side menu and upgrade classes.

`#include <button.hpp>`

Inheritance diagram for Button:



Public Member Functions

- **Button** (const std::string &label, sf::Vector2f size, sf::Vector2f position, sf::Color fill_color, sf::Color outline_↩ color, const sf::Font &font)
- void **center_text** ()

  Centers the sf::text object to the middle of the button.
- void set_outline_color (sf::Color outline_color)

  Sets outline color of the button box and color of the text.
- void set_fill_color (sf::Color fill_color)

  Sets fill color of the button box.
- void set_text_string (const std::string &label)

  Sets the button's text as the given string.
- void set_position (sf::Vector2f pos)

  Sets the position of the button background and centers the text according the button and text size.
- void set_size (sf::Vector2f size)

  Sets the size of the button.
- void set_font (const sf::Font &font)

  assigns the given reference to a font to the buttons sf::Text object.
- sf::Vector2f get_size ()

  Returns the buttons size.
- bool button_pressed ()

  Returns a true value if the button is pressed. Used in if-statements to trigger actions in menus.
- void reset_button ()

  Resets the button. Used after button_pressed in menus.
- bool is_mouse_over (sf::RenderWindow &window)

  Resets the button. Used after button_pressed in menus.
- void handle_events (sf::RenderWindow &window, const sf::Event &event, Level &lv)

  Handle mouse events given from the game class. Makes fill color brighter if mouse hovers on top of button. Makes button less transparent if mouse hovers on top of button (used as transparent in MainMenu and ChooseLevelMenu). Sets button pressed if mouse click is recorded on top of button.

Static Public Member Functions

- static bool inside_grid (sf::Vector2i mouse_pos, Level &lv)

  Checks if mouse position is inside grid.
- static std::pair< int, int > window_coords_to_grid_index (sf::Vector2i mouse_pos, Level &lv)

  calculates and returns grid index (column, row) of the level from the mouse position w.r.t window
- static Vector2D window_coords_to_level_coords (sf::Vector2i mouse_pos)

  calculates and returns Vector2D coords of the level from the mouse position w.r.t window

Protected Member Functions

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

  A function that overwrites the virtual function from sf::Drawable for drawing like window.draw(Button).

Protected Attributes

- sf::RectangleShape **_button**

  Background of the button.
- sf::Color **_button_fill_color**

  Fill color of the button.
- sf::Color **_button_outline_color**

  Outline color of the button.
- sf::Vector2f **_position**

  Position of the button.
- sf::Vector2f **_size**

  Size of the button.
- sf::Text **_text**

  Text obect of the button.
- bool **_button_pressed**

  The state of the button.

8.6.1 Detailed Description

A class to represent buttons in menus. Inherits from the sfml Drawable class to hide draw calls. Works as a transparent button in the two first menus and as an opaque button for side menu and upgrade classes.

Parameters

label	what to write in the button
size	size of the button
position	position of the button
fill_color	color of the button
outline_color	outline color of the button
font	a reference to the games font from resource handler

8.6.2 Member Function Documentation

8.6.2.1 button_pressed()

```
bool Button::button_pressed ( )
```

Returns a true value if the button is pressed. Used in if-statements to trigger actions in menus.

Returns

bool

8.6.2.2 draw()

```
void Button::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [protected], [virtual]
```

A function that overwrites the virtual function from sf::Drawable for drawing like window.draw(Button).

Parameters

target	Target to draw to
states	not used

Reimplemented in TowerDragButton.

8.6.2.3 get_size()

```
sf::Vector2f Button::get_size ( )
```

Returns the buttons size.

Returns

sf::Vector2f

8.6.2.4 handle_events()

```
void Button::handle_events (
            sf::RenderWindow & window,
            const sf::Event & event,
            Level & lv )
```

Handle mouse events given from the game class. Makes fill color brighter if mouse hovers on top of button. Makes button less transparent if mouse hovers on top of button (used as transparent in MainMenu and ChooseLevelMenu). Sets button pressed if mouse click is recorded on top of button.

Parameters

window	reference to the Game class's render window
event	reference to the Game classes event
lv	reference to the Game classes level class

8.6.2.5 inside_grid()

```
bool Button::inside_grid (
            sf::Vector2i mouse_pos,
            Level & lv )  [static]
```

Checks if mouse position is inside grid.

Parameters

mouse_pos	position of the mouse w.r.t window
lv	reference to the Game classes level class

Returns

bool

8.6.2.6 is_mouse_over()

```
bool Button::is_mouse_over (
            sf::RenderWindow & window )
```

Resets the button. Used after button_pressed in menus.

Returns

bool

8.6.2.7 reset_button()

```
void Button::reset_button ( )
```

Resets the button. Used after button_pressed in menus.

Returns

bool

8.6.2.8 set_fill_color()

```
void Button::set_fill_color (
            sf::Color fill_color )
```

Sets fill color of the button box.

Parameters

fill_color	color of the outline

8.6.2.9 set_font()

```
void Button::set_font (
            const sf::Font & font )
```

assigns the given reference to a font to the buttons sf::Text object.

Parameters

font	

8.6.2.10 set_outline_color()

```
void Button::set_outline_color (
            sf::Color outline_color )
```

Sets outline color of the button box and color of the text.

Parameters

outline_color	color of the outline

8.6.2.11 set_position()

```
void Button::set_position (
            sf::Vector2f pos )
```

Sets the position of the button background and centers the text according the button and text size.

Parameters

pos	

8.6.2.12 set_size()

```
void Button::set_size (
            sf::Vector2f size )
```

Sets the size of the button.

Parameters

size	

8.6.2.13 set_text_string()

```
void Button::set_text_string (
            const std::string & label )
```

Sets the button's text as the given string.

Parameters

label	

8.6.2.14 window_coords_to_grid_index()

```
std::pair< int, int > Button::window_coords_to_grid_index (
            sf::Vector2i mouse_pos,
            Level & lv )  [static]
```

calculates and returns grid index (column, row) of the level from the mouse position w.r.t window

Parameters

mouse_pos	position of the mouse w.r.t window
lv	reference to the Game classes level class

Returns

std::pair<int, int>

8.6.2.15 window_coords_to_level_coords()

```
Vector2D Button::window_coords_to_level_coords (
            sf::Vector2i mouse_pos )  [static]
```

calculates and returns Vector2D coords of the level from the mouse position w.r.t window

Parameters

mouse_pos	position of the mouse w.r.t window
lv	reference to the Game classes level class

Returns

Vector2D

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

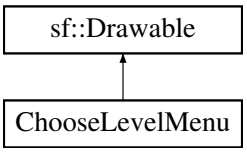
- src/button.hpp
- src/button.cpp

8.7 ChooseLevelMenu Class Reference

a class representing the level selection menu.

`#include <choose_level_menu.hpp>`

Inheritance diagram for ChooseLevelMenu:



Public Member Functions

- **ChooseLevelMenu** (ResourceHandler &rh, Level &level)
- **ChooseLevelMenu** (const ChooseLevelMenu &)=delete
- ChooseLevelMenu & **operator=** (const ChooseLevelMenu &)=delete
- void **disable_menu** ()

  Disables menu.
- void **enable_menu** ()

  Enables menu.
- int get_state ()

  Get the GameState to check the transition in Game class: 1 = stay on the menu 2 = load level in transtion and start game with pause.
- void **reset** ()

  Resets state and buttons.
- void handle_events (sf::RenderWindow &window, sf::Event &event)

  Forwards the events from window to the buttons.
- const std::string get_level_to_load ()

  Get the level to load for level.load_file()

8.7.1 Detailed Description

a class representing the level selection menu.

Parameters

ResourceHandler&	
Level&	

8.7.2 Member Function Documentation

8.7.2.1 get_level_to_load()

```
const std::string ChooseLevelMenu::get_level_to_load ( )
```

Get the level to load for level.load_file()

Returns

const std::string

8.7.2.2 get_state()

```
int ChooseLevelMenu::get_state ( )
```

Get the GameState to check the transition in Game class: 1 = stay on the menu 2 = load level in transtion and start game with pause.

Returns

int

8.7.2.3 handle_events()

```
void ChooseLevelMenu::handle_events (
            sf::RenderWindow & window,
            sf::Event & event )
```

Forwards the events from window to the buttons.

Parameters

window	
event	

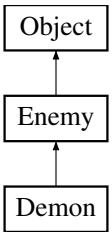
The documentation for this class was generated from the following files:

- src/choose_level_menu.hpp
- src/choose_level_menu.cpp

8.8 Demon Class Reference

```
#include <demon_enemy.hpp>
```

Inheritance diagram for Demon:



Public Member Functions

- Demon (Level &level, Vector2D &position, int health=150, int damage=10, int range=130, int attack_↩ speed=20, int type=ObjectTypes::NoobDemon_CanAttack, int speed=2, int defense=5, int size=2)

  Constructor to initialize a Demon Enemy object.
- ∼**Demon** ()

  Destructor for the Demon Enemy.
- bool attack ()

  Perform an attack action specific to the Demon Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ∼**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.8.1 Detailed Description

Class for a Demon enemy

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Demon()

```
Demon::Demon (
            Level & level,
            Vector2D & position,
            int health = 150,
            int damage = 10,
            int range = 130,
            int attack_speed = 20,
            int type = ObjectTypes::NoobDemon_CanAttack,
            int speed = 2,
            int defense = 5,
            int size = 2 )
```

Constructor to initialize a Demon Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Demon Enemy (Vector2D).
health	Initial health points of the Demon Enemy.
damage	Damage inflicted by the Demon Enemy.
range	Range of attack for the Demon Enemy.
attack_speed	Speed of attack for the Demon Enemy.
type	Type of the Demon Enemy.
speed	Speed attribute of the Demon Enemy.
defense	Defense attribute of the Demon Enemy.
size	Size of the Demon Enemy.

8.8.3 Member Function Documentation

8.8.3.1 attack()

```
bool Demon::attack ( )  [virtual]
```

Perform an attack action specific to the Demon Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

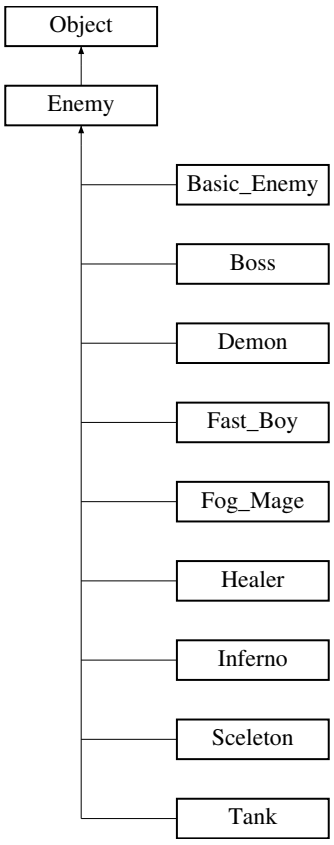
- src/demon_enemy.hpp
- src/demon_enemy.cpp

8.9 Enemy Class Reference

Enemy class for handling all enemies.

```
#include <enemy.hpp>
```

Inheritance diagram for Enemy:



Public Member Functions

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.
- virtual bool attack ()

  Virtual method for Object's attack action.

8.9.1 Detailed Description

Enemy class for handling all enemies.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Enemy()

```
Enemy::Enemy (
            Level & level,
            Vector2D & position,
            int health,
            int damage,
            int range,
            int attack_speed,
            int type,
            int speed,
            int defense,
            int size )
```

Constructor to initialize an Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Enemy (Vector2D).
health	Initial health points of the Enemy.
damage	Damage inflicted by the Enemy.
range	Range of attack for the Enemy.
attack_speed	Attack speed for the Enemy.
type	Type of the Enemy.
speed	Speed attribute of the Enemy.
defense	Defense attribute of the Enemy.
size	Size of the Enemy.

8.9.3 Member Function Documentation

8.9.3.1 get_defense()

```
int Enemy::get_defense ( ) const
```

Get the defense attribute of the Enemy.

Returns

Integer representing the defense attribute.

8.9.3.2 get_original_speed()

```
int Enemy::get_original_speed ( ) const
```

Get the original speed attribute of the Enemy.

Returns

Integer representing the original speed attribute.

8.9.3.3 get_prev_pos()

```
Vector2D Enemy::get_prev_pos ( )
```

Get the previous position of the Enemy.

Returns

Vector2D representing the previous position of the Enemy.

8.9.3.4 get_route()

```
std::vector< Vector2D > Enemy::get_route ( ) const
```

Get the route of the Enemy.

Returns

Vector of Vector2D representing the route of the Enemy.

8.9.3.5 get_size()

```
int Enemy::get_size ( ) const
```

Get the size of the Enemy.

Returns

Integer representing the size of the Enemy.

8.9.3.6 get_speed()

```
int Enemy::get_speed ( ) const
```

Get the speed attribute of the Enemy.

Returns

Integer representing the speed attribute.

8.9.3.7 lose_speed()

```
void Enemy::lose_speed (
            int amount )
```

Decrease the speed attribute of the Enemy.

Parameters

amount	Amount by which the speed is decreased.

8.9.3.8 set_prev_pos()

```
void Enemy::set_prev_pos (
            Vector2D pos )
```

Set the previous position of the Enemy.

Parameters

pos	New previous position to be set (Vector2D).

8.9.3.9 set_route_position()

```
void Enemy::set_route_position (
            Vector2D position )
```

Set the route position for the Enemy.

Parameters

position	New position to be set on the route (Vector2D).

8.9.3.10 set_speed()

```
void Enemy::set_speed (
            int amount )
```

Set the speed attribute of the Enemy.

Parameters

amount	New value for the speed attribute.

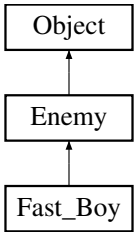
The documentation for this class was generated from the following files:

- src/enemy.hpp
- src/enemy.cpp

8.10 Fast_Boy Class Reference

```
#include <fastboy_enemy.hpp>
```

Inheritance diagram for Fast_Boy:



Public Member Functions

- Fast_Boy (Level &level, Vector2D &position, int health=10, int damage=0, int range=0, int attack_speed=1, int type=ObjectTypes::FastBoy, int speed=10, int defense=5, int size=2)

  Constructor to initialize a Fast Boy Enemy object.
- ~**Fast_Boy** ()

  Destructor for the Fast Boy Enemy.
- bool attack ()

  Perform an attack action specific to the Fast Boy Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.10.1 Detailed Description

Class for a Fast boy enemy

8.10.2 Constructor & Destructor Documentation

8.10.2.1 Fast_Boy()

```
Fast_Boy::Fast_Boy (
            Level & level,
            Vector2D & position,
            int health = 10,
            int damage = 0,
            int range = 0,
            int attack_speed = 1,
            int type = ObjectTypes::FastBoy,
            int speed = 10,
            int defense = 5,
            int size = 2 )
```

Constructor to initialize a Fast Boy Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Fast Boy Enemy (Vector2D).
health	Initial health points of the Fast Boy Enemy.
damage	Damage inflicted by the Fast Boy Enemy.
range	Range of attack for the Fast Boy Enemy.
attack_speed	Speed of attack for the Fast Boy Enemy.
type	Type of the Fast Boy Enemy.
speed	Speed attribute of the Fast Boy Enemy.
defense	Defense attribute of the Fast Boy Enemy.
size	Size of the Fast Boy Enemy.

8.10.3 Member Function Documentation

8.10.3.1 attack()

```
bool Fast_Boy::attack ( )  [virtual]
```

Perform an attack action specific to the Fast Boy Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

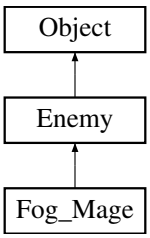
- src/fastboy_enemy.hpp
- src/fastboy_enemy.cpp

8.11 Fog_Mage Class Reference

Class for a Fog Mage enemy.

```
#include <fogmage_enemy.hpp>
```

Inheritance diagram for Fog_Mage:



Public Member Functions

- Fog_Mage (Level &level, Vector2D &position, int health=300, int damage=0, int range=150, int attack_↩ speed=50, int type=ObjectTypes::FogMage, int speed=1, int defense=10, int size=4)

  Constructor to initialize a Fog Mage Enemy object.
- ~**Fog_Mage** ()

  Destructor for the Fog Mage Enemy.
- bool attack ()

  Perform an attack action specific to the Fog Mage Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.11.1 Detailed Description

Class for a Fog Mage enemy.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 Fog_Mage()

```
Fog_Mage::Fog_Mage (
            Level & level,
            Vector2D & position,
            int health = 300,
            int damage = 0,
            int range = 150,
            int attack_speed = 50,
            int type = ObjectTypes::FogMage,
            int speed = 1,
            int defense = 10,
            int size = 4 )
```

Constructor to initialize a Fog Mage Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Fog Mage Enemy (Vector2D).
health	Initial health points of the Fog Mage Enemy.
damage	Damage inflicted by the Fog Mage Enemy.
range	Range of attack for the Fog Mage Enemy.
attack_speed	Speed of attack for the Fog Mage Enemy.
type	Type of the Fog Mage Enemy.
speed	Speed attribute of the Fog Mage Enemy.
defense	Defense attribute of the Fog Mage Enemy.
size	Size of the Fog Mage Enemy.

8.11.3 Member Function Documentation

8.11.3.1 attack()

```
bool Fog_Mage::attack ( )  [virtual]
```

Perform an attack action specific to the Fog Mage Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

- src/fogmage_enemy.hpp
- src/fogmage_enemy.cpp

8.12 Game Class Reference

A class for running the game. Opens a window in which a game loop handles user inputs, updates game state and draws game entities.

`#include <game.hpp>`

Public Member Functions

- **Game** ()

  Construct a new Game object.
- ~**Game** ()

  Destroy the Game object.
- **Game** (const Game &)=delete
- Game **operator=** (const Game &)=delete
- int get_side_bar_width () const

  Get the side bar width.
- int get_game_resolution () const

  Get the games resolution.
- void **run** ()

  Starts the game that loop until window is closed.

8.12.1 Detailed Description

A class for running the game. Opens a window in which a game loop handles user inputs, updates game state and draws game entities.

8.12.2 Member Function Documentation

8.12.2.1 get_game_resolution()

```
int Game::get_game_resolution ( ) const
```

Get the games resolution.

Returns

int

8.12.2.2 get_side_bar_width()

```
int Game::get_side_bar_width ( ) const
```

Get the side bar width.

Returns

int

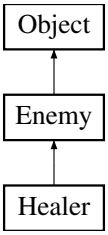
The documentation for this class was generated from the following files:

- src/game.hpp
- src/game.cpp

8.13 Healer Class Reference

`#include <healer_enemy.hpp>`

Inheritance diagram for Healer:



Public Member Functions

- Healer (Level &level, Vector2D &position, int health=200, int damage=0, int range=100, int attack_speed=20, int type=ObjectTypes::HealerPriest, int speed=1, int defense=10, int size=2)

  Constructor to initialize a Healer Enemy object.
- ~**Healer** ()

  Destructor for the Healer Enemy.
- bool attack ()

  Perform an attack action specific to the Healer Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.13.1 Detailed Description

Class for a Healer enemy

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Healer()

```
Healer::Healer (
            Level & level,
            Vector2D & position,
            int health = 200,
            int damage = 0,
            int range = 100,
            int attack_speed = 20,
            int type = ObjectTypes::HealerPriest,
            int speed = 1,
            int defense = 10,
            int size = 2 )
```

Constructor to initialize a Healer Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Healer Enemy (Vector2D).
health	Initial health points of the Healer Enemy.
damage	Damage inflicted by the Healer Enemy.
range	Range of attack for the Healer Enemy.
attack_speed	Speed of attack for the Healer Enemy.
type	Type of the Healer Enemy.
speed	Speed attribute of the Healer Enemy.
defense	Defense attribute of the Healer Enemy.
size	Size of the Healer Enemy.

8.13.3 Member Function Documentation

8.13.3.1 attack()

```
bool Healer::attack ( )  [virtual]
```

Perform an attack action specific to the Healer Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

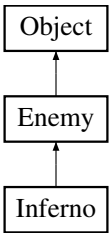
The documentation for this class was generated from the following files:

- src/healer_enemy.hpp
- src/healer_enemy.cpp

8.14 Inferno Class Reference

```
#include <inferno_enemy.hpp>
```

Inheritance diagram for Inferno:



Public Member Functions

- Inferno (Level &level, Vector2D &position, int health=400, int damage=40, int range=150, int attack_↩ speed=60, int type=ObjectTypes::InfernoMage, int speed=3, int defense=10, int size=4)

  Constructor to initialize an Inferno Enemy object.
- ∼**Inferno** ()

  Destructor for the Inferno Enemy.
- bool attack ()

  Perform an attack action specific to the Inferno Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.14.1 Detailed Description

Class for an Inferno enemy

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Inferno()

```
Inferno::Inferno (
            Level & level,
            Vector2D & position,
            int health = 400,
            int damage = 40,
            int range = 150,
            int attack_speed = 60,
            int type = ObjectTypes::InfernoMage,
            int speed = 3,
            int defense = 10,
            int size = 4 )
```

Constructor to initialize an Inferno Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Inferno Enemy (Vector2D).
health	Initial health points of the Inferno Enemy.
damage	Damage inflicted by the Inferno Enemy.
range	Range of attack for the Inferno Enemy.
attack_speed	Speed of attack for the Inferno Enemy.
type	Type of the Inferno Enemy.
speed	Speed attribute of the Inferno Enemy.
defense	Defense attribute of the Inferno Enemy.
size	Size of the Inferno Enemy.

8.14.3 Member Function Documentation

8.14.3.1 attack()

```
bool Inferno::attack ( )  [virtual]
```

Perform an attack action specific to the Inferno Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

- src/inferno_enemy.hpp
- src/inferno_enemy.cpp

8.15 Level Class Reference

Class that controls level of the game.

`#include <level.hpp>`

Public Member Functions

- Level (int resolution, int cash, int lives)
  Construct a new Level object.
- ~**Level** ()
  Destroy the Level object.
- int get_round () const
  Get current round.
- int get_cash () const
  Get current money situation.
- int get_lives () const
  Get current lives.
- std::vector< std::vector< Square * > > get_grid () const
  Get current the grid.
- int get_square_size () const
  Get the size of each square.
- void **make_grid** ()
  Makes new grid.
- void **plus_round** ()
- void add_cash (int how_much)
  Add money.
- void take_cash (int how_much)
  Take money from player.
- void take_lives (int how_much)
  Take lives from player.
- void add_lives (int how_much)
  Add lives.
- void reset (int start_cash, int start_lives)
  resets the level for a new game.
- std::vector< Enemy * > get_enemies () const
  Get the vector of enemies.
- bool add_enemy (Enemy *enemy)
  Adds pointer of enemy to vector of all enemies.
- bool remove_enemy (Enemy *enemy)
  Removes enemy from vector.
- void **remove_all_enemies** ()
  empties and deletes enemies from vector of all enemies
- bool add_enemy_by_type (int type, Vector2D pos)
  Makes and adds new enemy by type and position of enemy.
- std::vector< Tower * > get_towers () const
  Get the vector of towers.
- bool add_tower (Tower *tower)
  Adds tower to vector of all towers.
- bool remove_tower (Tower *tower)

Removes tower from vector of all towers.

- void **remove_all_towers** ()

  empties and deletes towers from vector of all towers

- bool add_tower_by_type (int type, Vector2D pos)

  Makes and adds new tower to vector of all towers.

- std::pair< int, int > current_row_col (Object *obj)

  Returns current column and row of object.

- Square * current_square (Object *obj)

  Returns pointer to current square of object.

- Square * get_square_by_pos (Vector2D pos)

  Returns pointer to square by position.

- std::vector< Direction > next_road (Enemy *enemy)

  Returns vector, where road continues from current square.

- void **print_objects** ()

  Print all objects.

- int read_file (const std::string &file_name)

  Load level from file.

- int save_to_file (const std::string &file_name)

  Saves current level to file.

- void **print_map** ()

  Prints out current map.

- std::pair< int, int > can_go_notstart (Direction dir, std::vector< Direction > prev_dirs, int row, int col, bool can_go_left)

  Helper functions for randomly generate Handles situation where it's not few of first round.

- std::pair< int, int > can_go_start (Direction dir, std::vector< Direction > dir_list, int row, int col)

  Helper functions for randomly generate Handles situation where its first few moves.

- bool randomly_generate ()

  Creates fully random level.

- Square * get_first_road ()

  Returns first peace of the road Helps enemies to spawn in right place.

8.15.1 Detailed Description

Class that controls level of the game.

Class holds current grid, all enemies and tower. Also it keeps track of round, money and live situation Level can load hard coded maps from files, save current levels to files or randomly generate fully new one

Parameters

resolution	of window
cash	starting cash
lives	starting lives

8.15.2 Constructor & Destructor Documentation

8.15.2.1 Level()

```
Level::Level (
```

```
            int resolution,
            int cash,
            int lives )
```

Construct a new Level object.

Parameters

resolution	
cash	
lives	

8.15.3 Member Function Documentation

8.15.3.1 add_cash()

```
void Level::add_cash (
            int how_much )
```

Add money.

Parameters

how_much	How much money want to be added

8.15.3.2 add_enemy()

```
bool Level::add_enemy (
            Enemy * enemy )
```

Adds pointer of enemy to vector of all enemies.

Parameters

enemy	Pointer to enemy

Returns

true if is added

false if not added

8.15.3.3 add_enemy_by_type()

```
bool Level::add_enemy_by_type (
            int type,
            Vector2D pos )
```

Makes and adds new enemy by type and position of enemy.

Parameters

type	type of enemy
pos	position of enemy

Returns

true if added

false if not added

8.15.3.4 add_lives()

```
void Level::add_lives (
            int how_much )
```

Add lives.

Parameters

how_much	How much lives is added

8.15.3.5 add_tower()

```
bool Level::add_tower (
            Tower * tower )
```

Adds tower to vector of all towers.

Parameters

tower	Pointer to tower

Returns

true if added

false if not added

8.15.3.6 add_tower_by_type()

```
bool Level::add_tower_by_type (
            int type,
            Vector2D pos )
```

Makes and adds new tower to vector of all towers.

Parameters

type	type of tower
pos	position of tower

Returns

true if added

false if not added

8.15.3.7 can_go_notstart()

```
std::pair< int, int > Level::can_go_notstart (
            Direction dir,
            std::vector< Direction > prev_dirs,
            int row,
            int col,
            bool can_go_left )
```

Helper functions for randomly generate Handles situation where it's not few of first round.

Parameters

dir	Direction where randomly generate wants to go
prev_dirs	Vector of all previous directions
row	Current row
col	Current col
can_go_left	Restricts how many time function can go left

Returns

std::pair<int, int> Returns pair of next row and column

8.15.3.8 can_go_start()

```
std::pair< int, int > Level::can_go_start (
            Direction dir,
            std::vector< Direction > dir_list,
            int row,
            int col )
```

Helper functions for randomly generate Handles situation where its first few moves.

Parameters

dir	Direction where randomly generate wants to go
prev_dirs	Vector of all previous directions
row	Current row
col	Current col

Returns

std::pair<int, int> Returns pair of next row and column

8.15.3.9 current_row_col()

```
std::pair< int, int > Level::current_row_col (
            Object * obj )
```

Returns current column and row of object.

Parameters

obj	Pointer to object

Returns

std::pair<int, int> => <col, row>

8.15.3.10 current_square()

```
Square * Level::current_square (
            Object * obj )
```

Returns pointer to current square of object.

Parameters

obj	Pointer to object

Returns

Pointer to square

8.15.3.11 get_cash()

```
int Level::get_cash ( ) const
```

Get current money situation.

Returns

int

8.15.3.12 get_enemies()

```
std::vector< Enemy * > Level::get_enemies ( ) const
```

Get the vector of enemies.

Returns

std::vector<Enemy*>

8.15.3.13 get_first_road()

```
Square * Level::get_first_road ( )
```

Returns first peace of the road Helps enemies to spawn in right place.

Returns

Square* Pointer of square where first road is

8.15.3.14 get_grid()

```
std::vector< std::vector< Square * > > Level::get_grid ( ) const
```

Get current the grid.

Returns

std::vector<std::vector<Square*>>

8.15.3.15 get_lives()

```
int Level::get_lives ( ) const
```

Get current lives.

Returns

int

8.15.3.16 get_round()

```
int Level::get_round ( ) const
```

Get current round.

Returns

int

8.15.3.17 get_square_by_pos()

```
Square * Level::get_square_by_pos (
            Vector2D pos )
```

Returns pointer to square by position.

Parameters

pos	Position on map

Returns

Pointer to square

8.15.3.18 get_square_size()

```
int Level::get_square_size ( ) const
```

Get the size of each square.

Returns

int

8.15.3.19 get_towers()

```
std::vector< Tower * > Level::get_towers ( ) const
```

Get the vector of towers.

Returns

std::vector<Tower*>

8.15.3.20 next_road()

```
std::vector< Direction > Level::next_road (
            Enemy * enemy )
```

Returns vector, where road continues from current square.

Parameters

enemy	Pointer to enemy

Returns

Vector of all directions where enemy can go

8.15.3.21 randomly_generate()

```
bool Level::randomly_generate ( )
```

Creates fully random level.

Returns

true If it was successful

false If it wasn't

8.15.3.22 read_file()

```
int Level::read_file (
            const std::string & file_name )
```

Load level from file.

Parameters

file_name	Name of file from where map is loaded

Returns

int 1 if maps is loaded and -1 if load failed

8.15.3.23 remove_enemy()

```
bool Level::remove_enemy (
            Enemy * enemy )
```

Removes enemy from vector.

Parameters

enemy	pointer to enemy

Returns

true if removed

false if not removes

8.15.3.24 remove_tower()

```
bool Level::remove_tower (
            Tower * tower )
```

Removes tower from vector of all towers.

Parameters

tower	Pointer to tower

Returns

true if removed

false if not removed

8.15.3.25 reset()

```
void Level::reset (
            int start_cash,
            int start_lives )
```

resets the level for a new game.

Parameters

start_cash	cash the new game starts with
start_lives	lives the new game starts with

8.15.3.26 save_to_file()

```
int Level::save_to_file (
            const std::string & file_name )
```

Saves current level to file.

Parameters

file_name	Name of file where map is saved

Returns

int 1 if map is saved and -1 if failed

8.15.3.27 take_cash()

```
void Level::take_cash (
            int how_much )
```

Take money from player.

Parameters

how_much	How much money is taken

8.15.3.28 take_lives()

```
void Level::take_lives (
            int how_much )
```

Take lives from player.

Parameters

how_much	How much lives is taken

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

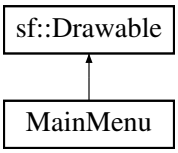
- src/level.hpp
- src/level.cpp

8.16 MainMenu Class Reference

A class representing the start menu / main menu of the game.

`#include <main_menu.hpp>`

Inheritance diagram for MainMenu:



Public Member Functions

- **MainMenu** (ResourceHandler &rh, Level &level)
- **MainMenu** (const MainMenu &)=delete
- MainMenu & **operator=** (const MainMenu &)=delete
- void **disable_menu** ()

  Disables menu.
- void **enable_menu** ()

  Enables menu.
- int get_state ()

  Get the GameState to check the transition in Game class: 0 = stay on the menu, 1 = go to choose level menu, or 2 = generate random level and start pause.
- void **reset** ()

  Resets state and buttons after end screen.
- void handle_events (sf::RenderWindow &window, sf::Event &event)

  Forward the events from window to the buttons.

8.16.1 Detailed Description

A class representing the start menu / main menu of the game.

Parameters

ResourceHandler&	
Level&	

8.16.2 Member Function Documentation

8.16.2.1 get_state()

```
int MainMenu::get_state ( )
```

Get the GameState to check the transition in Game class: 0 = stay on the menu, 1 = go to choose level menu, or 2 = generate random level and start pause.

Returns

int

8.16.2.2 handle_events()

```
void MainMenu::handle_events (
            sf::RenderWindow & window,
            sf::Event & event )
```

Forward the events from window to the buttons.

Parameters

window	
event	

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

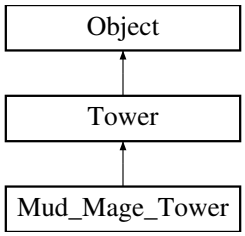
- src/main_menu.hpp
- src/main_menu.cpp

8.17 Mud_Mage_Tower Class Reference

Class for a Mud Mage tower.

```
#include <mud_mage_tower.hpp>
```

Inheritance diagram for Mud_Mage_Tower:



Public Member Functions

- Mud_Mage_Tower (Level &current_level, Vector2D &position, int health=400, int damage=30, int range=180, int attack_speed=30, int type=ObjectTypes::MudMageTower, int price=200, int level=1)

  Constructor to initialize a Mud Mage Tower object.
- ~**Mud_Mage_Tower** ()

  Destructor for the Mud Mage Tower.
- bool attack ()

  Perform an attack action specific to the Mud Mage Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

Get the position of the object.

- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.17.1 Detailed Description

Class for a Mud Mage tower.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 Mud_Mage_Tower()

```
Mud_Mage_Tower::Mud_Mage_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 400,
            int damage = 30,
            int range = 180,
            int attack_speed = 30,
            int type = ObjectTypes::MudMageTower,
            int price = 200,
            int level = 1 )
```

Constructor to initialize a Mud Mage Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Mud Mage Tower (Vector2D).
health	Initial health points of the Mud Mage Tower.
damage	Damage inflicted by the Mud Mage Tower.
range	Range of attack for the Mud Mage Tower.
attack_speed	Speed of attack for the Mud Mage Tower.
type	Type of the Mud Mage Tower.
price	Price of the Mud Mage Tower.
level	Level of the Mud Mage Tower.

8.17.3 Member Function Documentation

8.17.3.1 attack()

```
bool Mud_Mage_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Mud Mage Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

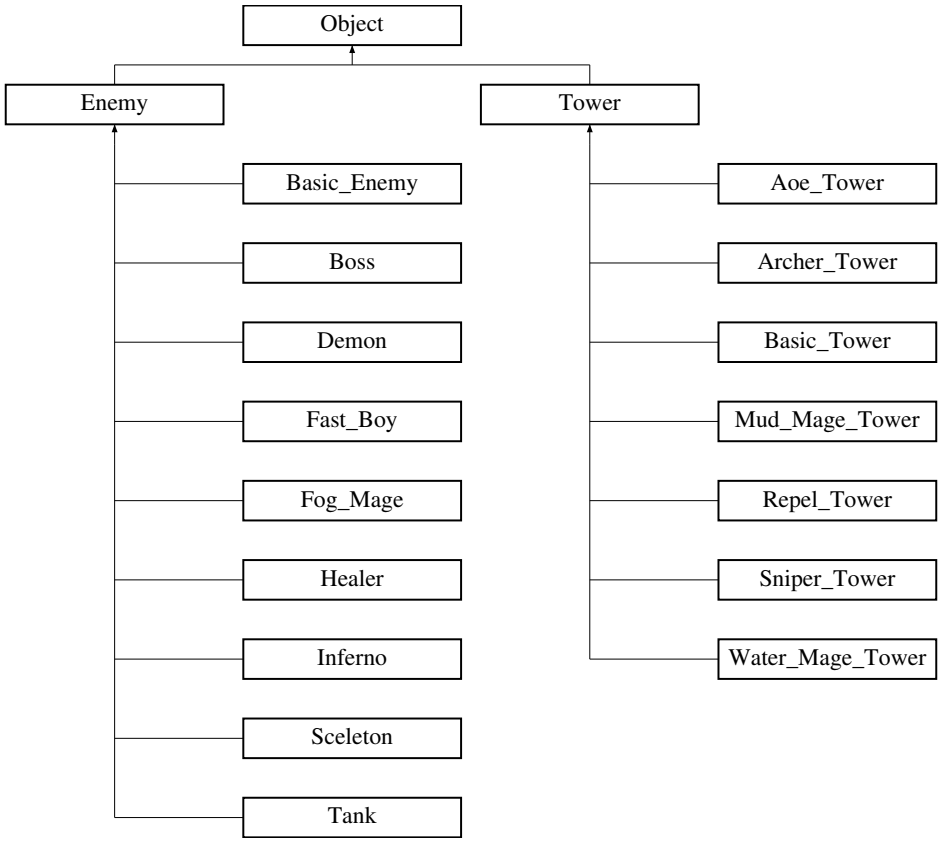
- src/mud_mage_tower.hpp
- src/mud_mage_tower.cpp

8.18 Object Class Reference

Class that defines the behavior of all objects in the game.

`#include <object.hpp>`

Inheritance diagram for Object:



Public Member Functions

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.

- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.
- virtual bool attack ()

  Virtual method for Object's attack action.

8.18.1 Detailed Description

Class that defines the behavior of all objects in the game.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 Object()

```
Object::Object (
            Level & level,
            Vector2D & position,
            int health,
            int damage,
            int range,
            int attack_speed,
            int type )
```

Constructor to initialize an Object.

Parameters

level	Reference to the Level object.
position	Initial position of the Object (Vector2D).
health	Initial health points of the Object.
damage	Damage inflicted by the Object.
range	Range of attack for the Object.
attack_speed	Speed of attack for the Object.
type	Type of the Object.

8.18.3 Member Function Documentation

8.18.3.1 attack()

```
bool Object::attack ( )  [virtual]
```

Virtual method for Object's attack action.

Returns

true if the attack is successful, false otherwise.

Reimplemented in Aoe_Tower, Archer_Tower, Basic_Enemy, Basic_Tower, Boss, Demon, Fast_Boy, Fog_Mage, Healer, Inferno, Mud_Mage_Tower, Repel_Tower, Sceleton, Sniper_Tower, Tank, and Water_Mage_Tower.

8.18.3.2 distance_to()

```
double Object::distance_to (
            const Vector2D & target_position )
```

Calculate the distance between the object and a target position.

Parameters

target_position	The position of the target (Vector2D).

Returns

Double representing the distance between the object and the target.

8.18.3.3 gain_attack_speed()

```
void Object::gain_attack_speed (
            int amount )
```

Increase the attack speed of the object.

Parameters

amount	Amount by which the attack speed is increased.

8.18.3.4 gain_damage()

```
void Object::gain_damage (
            int amount )
```

Increase the damage value of the object.

Parameters

amount	Amount by which the damage is increased.

8.18.3.5 gain_health()

```
void Object::gain_health (
            int amount )
```

Increase the health value of the object.

Parameters

amount	Amount by which the health is increased.

8.18.3.6 gain_range()

```
void Object::gain_range (
            int amount )
```

Increase the attack range of the object.

Parameters

amount	Amount by which the attack range is increased.

8.18.3.7 get_attack_counter()

```
int Object::get_attack_counter ( ) const
```

Get the attack counter value of the object.

Returns

Integer representing the attack counter value.

8.18.3.8 get_attack_speed()

```
int Object::get_attack_speed ( ) const
```

Get the attack speed of the object.

Returns

Integer representing the attack speed.

8.18.3.9 get_damage()

```
int Object::get_damage ( ) const
```

Get the damage value of the object.

Returns

Integer representing the damage value.

8.18.3.10 get_health()

```
int Object::get_health ( ) const
```

Get the health value of the object.

Returns

Integer representing the health value.

8.18.3.11 get_level_reference()

```
Level & Object::get_level_reference ( ) const
```

Get a reference to the Level object associated with this object.

Returns

Reference to the Level object.

8.18.3.12 get_original_attack_speed()

```
int Object::get_original_attack_speed ( ) const
```

Get the original attack speed of the object.

Returns

Integer representing the original attack speed.

8.18.3.13 get_position()

```
const Vector2D Object::get_position ( ) const
```

Get the position of the object.

Returns

Vector2D object representing the position.

8.18.3.14 get_range()

```
int Object::get_range ( ) const
```

Get the attack range of the object.

Returns

Integer representing the attack range.

8.18.3.15 get_reset_counter()

```
int Object::get_reset_counter ( ) const
```

Get the reset counter value of the object.

Returns

Integer representing the reset counter value.

8.18.3.16 get_state()

```
State Object::get_state ( )
```

Get the current state of the object.

Returns

State enum representing the current state.

8.18.3.17 get_type()

```
int Object::get_type ( ) const
```

Get the type of the object.

Returns

Integer representing the type of the object.

8.18.3.18 get_wait_time()

```
int Object::get_wait_time ( ) const
```

Get the reset wait time for all objects.

Returns

Integer the amount of time to wait.

8.18.3.19 lose_attack_speed()

```
void Object::lose_attack_speed (
            int amount )
```

Decrease the attack speed of the object.

Parameters

amount	Amount by which the attack speed is decreased.

8.18.3.20 lose_health()

```
void Object::lose_health (
            int amount )
```

Decrease the health value of the object.

Parameters

amount	Amount by which the health is decreased.

8.18.3.21 set_attack_counter()

```
void Object::set_attack_counter (
            const int amount )
```

Set the attack counter value of the object.

Parameters

amount	New value for the attack counter.

8.18.3.22 set_attack_speed()

```
void Object::set_attack_speed (
            const int amount )
```

Set the attack speed of the object.

Parameters

amount	New value for the attack speed.

8.18.3.23 set_original_attack_speed()

```
void Object::set_original_attack_speed (
            const int amount )
```

Set the original attack speed of the object.

Parameters

amount	New value for the original attack speed.

8.18.3.24 set_position()

```
void Object::set_position (
            const Vector2D & position )
```

Set the position of the object.

Parameters

position	New position of the object (Vector2D).

8.18.3.25 set_reset_counter()

```
void Object::set_reset_counter (
            const int amount )
```

Set the reset counter value of the object.

Parameters

amount	New value for the reset counter.

8.18.3.26 set_state()

```
void Object::set_state (
            State state )
```

Set the state of the object.

Parameters

state	The new state of the object.

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

- src/object.hpp
- src/object.cpp

8.19 Renderer Class Reference

Class for creating drawable game objects for window.draw([DRAWABLE GAME OBJECT])

```
#include <renderer.hpp>
```

Public Member Functions

- **Renderer** ()

  Construct a new Renderer object.
- **Renderer** (ResourceHandler &rh)

  Destroy the Renderer object.
- **Renderer** (const Renderer &)=delete
- Renderer **operator=** (const Renderer &)=delete

- void make_drawable_level (Level &lv)

  Makes displays current level.
- void **make_level_info_texts** (int game_resolution, int side_bar_width)
- void draw_level (sf::RenderWindow &rwindow)

  draw background
- void draw_enemy (sf::RenderWindow &rwindow, Enemy *e_ptr, int frame, int move_animation)

  draw single enemy
- void draw_enemies (sf::RenderWindow &rwindow, std::vector< Enemy * > enemies, int frame, int move_animation)

  draw enemies on from a list
- void draw_tower (sf::RenderWindow &rwindow, Tower *t_ptr, int frame)

  draw single enemy
- void draw_towers (sf::RenderWindow &rwindow, std::vector< Tower * > towers, int frame)

  draw towers on from a list
- void draw_end_screen_win (sf::RenderWindow &rwindow)

  victory screen on GameState::EndScreen
- void draw_end_screen_lose (sf::RenderWindow &rwindow)

  game over screen on GameState::EndScreen

8.19.1 Detailed Description

Class for creating drawable game objects for window.draw([DRAWABLE GAME OBJECT])

8.19.2 Member Function Documentation

8.19.2.1 draw_end_screen_lose()

```
void Renderer::draw_end_screen_lose (
            sf::RenderWindow & rwindow )
```

game over screen on GameState::EndScreen

Parameters

rwindow	window where to draw

8.19.2.2 draw_end_screen_win()

```
void Renderer::draw_end_screen_win (
            sf::RenderWindow & rwindow )
```

victory screen on GameState::EndScreen

Parameters

rwindow	window where to draw

8.19.2.3 draw_enemies()

```
void Renderer::draw_enemies (
            sf::RenderWindow & rwindow,
            std::vector< Enemy * > enemies,
            int frame,
            int move_animation )
```

draw enemies on from a list

Parameters

rwindow	window where to draw
enemies	list of enemies
frame	current frame of animation
move_animation	current frame for enemies move animation (for smoother movement)

8.19.2.4 draw_enemy()

```
void Renderer::draw_enemy (
            sf::RenderWindow & rwindow,
            Enemy * e_ptr,
            int frame,
            int move_animation )
```

draw single enemy

Parameters

rwindow	window where to draw
e_ptr	pointer to enemy
frame	current frame of animation
move_animation	current frame for enemies move animation (for smoother movement)

8.19.2.5 draw_level()

```
void Renderer::draw_level (
            sf::RenderWindow & rwindow )
```

draw background

Parameters

rwindow	windows where background is drawn

8.19.2.6 draw_tower()

```
void Renderer::draw_tower (
```

```
            sf::RenderWindow & rwindow,
            Tower * t_ptr,
            int frame )
```

draw single enemy

Parameters

rwindow	window where to draw
t_ptr	pointer to tower
frame	current frame of animation

8.19.2.7 draw_towers()

```
void Renderer::draw_towers (
            sf::RenderWindow & rwindow,
            std::vector< Tower * > towers,
            int frame )
```

draw towers on from a list

Parameters

rwindow	window where to draw
towers	list of towers
frame	current frame of animation

8.19.2.8 make_drawable_level()

```
void Renderer::make_drawable_level (
            Level & lv )
```

Makes displays current level.

Parameters

lv	level that is displayed

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

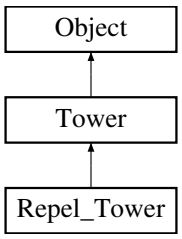
- src/renderer.hpp
- src/renderer.cpp

8.20 Repel_Tower Class Reference

Class for a Repel tower.

`#include <repel_tower.hpp>`

Inheritance diagram for Repel_Tower:



Public Member Functions

- Repel_Tower (Level &current_level, Vector2D &position, int health=350, int damage=0, int range=150, int attack_speed=50, int type=ObjectTypes::RepelMageTower, int price=250, int level=1)

  Constructor to initialize a Repel Tower object.
- ~**Repel_Tower** ()

  Destructor for the Repel Tower.
- bool attack ()

  Perform an attack action specific to the Repel Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

Get the attack speed of the object.

- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.20.1 Detailed Description

Class for a Repel tower.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 Repel_Tower()

```
Repel_Tower::Repel_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 350,
            int damage = 0,
            int range = 150,
            int attack_speed = 50,
            int type = ObjectTypes::RepelMageTower,
            int price = 250,
            int level = 1 )
```

Constructor to initialize a Repel Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Repel Tower (Vector2D).
health	Initial health points of the Repel Tower.
damage	Damage inflicted by the Repel Tower.
range	Range of attack for the Repel Tower.
attack_speed	Speed of attack for the Repel Tower.
type	Type of the Repel Tower.
price	Price of the Repel Tower.
level	Level of the Repel Tower.

8.20.3 Member Function Documentation

8.20.3.1 attack()

```
bool Repel_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Repel Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

- src/repel_tower.hpp
- src/repel_tower.cpp

8.21 ResourceHandler Class Reference

A class to load all resources at one place; loads textures, fonts, tower attributes for the menu. Loads everything when it is constructed, distributes resources as references to shared pointers.

`#include <resource_handler.hpp>`

Public Member Functions

- sf::Texture & get_texture_tower (int type)

  Get the texture of tower by ObjectType enum as reference.
- sf::Texture & get_texture_enemy (int type)

  Get the texture of enemy by ObjectType enum as reference.
- sf::Texture & get_texture_tile (int type)

  Get the texture of level square as reference. 0 = grass, 1 = road, 2 = house.
- sf::Texture & get_texture_menu (int type)

  Get the texture of menus as reference. 0 = start menu, 1 = level menu, 2 = side menu, 3 = game over, 4 = victory.
- sf::Font & get_font ()

  Get the sf::Font object as reference.
- int get_tower_info (int tower_type, int attr_type)

  Get the towers attribute with ObjectType and TowerAttribute.
- const std::string & get_tower_name (int type)

  Get the tower's name as const reference to a string.
- sf::Texture & get_texture_attribute (int type)

  Get the texture for attributes as a reference with TowerAttribute.

8.21.1 Detailed Description

A class to load all resources at one place; loads textures, fonts, tower attributes for the menu. Loads everything when it is constructed, distributes resources as references to shared pointers.

8.21.2 Member Function Documentation

8.21.2.1 get_font()

```
sf::Font & ResourceHandler::get_font ( )
```

Get the sf::Font object as reference.

Returns

sf::Font&

8.21.2.2 get_texture_attribute()

```
sf::Texture & ResourceHandler::get_texture_attribute (
            int type )
```

Get the texture for attributes as a reference with TowerAttribute.

Parameters

type	

Returns

sf::Texture&

8.21.2.3 get_texture_enemy()

```
sf::Texture & ResourceHandler::get_texture_enemy (
            int type )
```

Get the texture of enemy by ObjectType enum as reference.

Parameters

type	

Returns

sf::Texture&

8.21.2.4 get_texture_menu()

```
sf::Texture & ResourceHandler::get_texture_menu (
            int type )
```

Get the texture of menus as reference. 0 = start menu, 1 = level menu, 2 = side menu, 3 = game over, 4 = victory.

Parameters

type	

Returns

sf::Texture&

8.21.2.5 get_texture_tile()

```
sf::Texture & ResourceHandler::get_texture_tile (
            int type )
```

Get the texture of level square as reference. 0 = grass, 1 = road, 2 = house.

Parameters

type	

Returns

sf::Texture&

8.21.2.6 get_texture_tower()

```
sf::Texture & ResourceHandler::get_texture_tower (
            int type )
```

Get the texture of tower by ObjectType enum as reference.

Parameters

type	

Returns

sf::Texture&

8.21.2.7 get_tower_info()

```
int ResourceHandler::get_tower_info (
            int tower_type,
            int attr_type )
```

Get the towers attribute with ObjectType and TowerAttribute.

Parameters

tower_type	
attr_type	

Returns

int

8.21.2.8 get_tower_name()

```
const std::string & ResourceHandler::get_tower_name (
            int type )
```

Get the tower's name as const reference to a string.

Parameters

type	

Returns

const std::string&

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

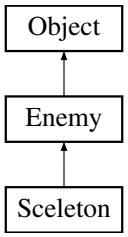
- src/resource_handler.hpp
- src/resource_handler.cpp

8.22 Sceleton Class Reference

Class for Skeleton enemy.

`#include <sceleton_enemy.hpp>`

Inheritance diagram for Sceleton:



Public Member Functions

- Sceleton (Level &level, Vector2D &position, int health=100, int damage=0, int range=0, int attack_speed=0, int type=ObjectTypes::NoobSkeleton_NoAttack, int speed=3, int defense=5, int size=2)

  Constructor to initialize a Sceleton Enemy object.
- ~**Sceleton** ()

  Destructor for the Sceleton Enemy.
- bool attack ()

  Perform an attack action specific to the Sceleton Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.

- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.22.1 Detailed Description

Class for Skeleton enemy.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 Sceleton()

```
Sceleton::Sceleton (
            Level & level,
            Vector2D & position,
            int health = 100,
            int damage = 0,
            int range = 0,
            int attack_speed = 0,
            int type = ObjectTypes::NoobSkeleton_NoAttack,
            int speed = 3,
            int defense = 5,
            int size = 2 )
```

Constructor to initialize a Sceleton Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Sceleton Enemy (Vector2D).
health	Initial health points of the Sceleton Enemy.
damage	Damage inflicted by the Sceleton Enemy.
range	Range of attack for the Sceleton Enemy.
attack_speed	Speed of attack for the Sceleton Enemy.
type	Type of the Sceleton Enemy.
speed	Speed attribute of the Sceleton Enemy.
defense	Defense attribute of the Sceleton Enemy.
size	Size of the Sceleton Enemy.

8.22.3 Member Function Documentation

8.22.3.1 attack()

```
bool Sceleton::attack ( )  [virtual]
```

Perform an attack action specific to the Sceleton Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

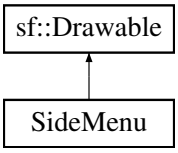
- src/sceleton_enemy.hpp
- src/sceleton_enemy.cpp

8.23 SideMenu Class Reference

Implements side menu of the game. Features:

`#include <side_menu.hpp>`

Inheritance diagram for SideMenu:



Public Member Functions

- **SideMenu** (float game_resolution, float sidebar_width, ResourceHandler &rh, Level &level)
- **SideMenu** (const SideMenu &)=delete
- SideMenu & **operator=** (const SideMenu &)=delete
- void **update** ()

  updates the texts and buttons according to the cash, lives, rounds of level
- void handle_events (sf::RenderWindow &window, const sf::Event &event)

  distributes the window and event references to the buttons.
- void **disable_buttons** ()

  disables handle events
- void **enable_buttons** ()

  enables handle events
- int get_state ()

  Get the the next game state, either continue in pause or if round button pressed transition to round(implemented in game class)
- void **pause** ()

  Sets state to GameState::Pause.

8.23.1 Detailed Description

Implements side menu of the game. Features:

- 6 drag buttons to purchase towers
- one button to start rounds
- displays the rounds, cash and lives stats

Parameters

game_resolution	
sidebar_width	
rh	
level	

8.23.2 Member Function Documentation

8.23.2.1 get_state()

```
int SideMenu::get_state ( )
```

Get the the next game state, either continue in pause or if round button pressed transition to round(implemented in game class)

Returns

int

8.23.2.2 handle_events()

```
void SideMenu::handle_events (
            sf::RenderWindow & window,
            const sf::Event & event )
```

distributes the window and event references to the buttons.

Parameters

window	
event	

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

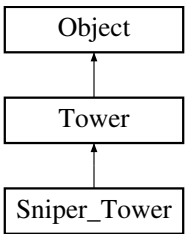
- src/side_menu.hpp
- src/side_menu.cpp

8.24 Sniper_Tower Class Reference

Class for Sniper tower.

```
#include <sniper_tower.hpp>
```

Inheritance diagram for Sniper_Tower:



Public Member Functions

- Sniper_Tower (Level &current_level, Vector2D &position, int health=200, int damage=150, int range=999, int attack_speed=120, int type=ObjectTypes::SniperTower, int price=200, int level=1)

  Constructor to initialize a Sniper Tower object.
- ~**Sniper_Tower** ()

  Destructor for the Sniper Tower.
- bool attack ()

  Perform an attack action specific to the Sniper Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.

- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.24.1 Detailed Description

Class for Sniper tower.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 Sniper_Tower()

```
Sniper_Tower::Sniper_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 200,
            int damage = 150,
            int range = 999,
```

```
int attack_speed = 120,
int type = ObjectTypes::SniperTower,
int price = 200,
int level = 1 )
```

Constructor to initialize a Sniper Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Sniper Tower (Vector2D).
health	Initial health points of the Sniper Tower.
damage	Damage inflicted by the Sniper Tower.
range	Range of attack for the Sniper Tower.
attack_speed	Speed of attack for the Sniper Tower.
type	Type of the Sniper Tower.
price	Price of the Sniper Tower.
level	Level of the Sniper Tower.

8.24.3 Member Function Documentation

8.24.3.1 attack()

```
bool Sniper_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Sniper Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

- src/sniper_tower.hpp
- src/sniper_tower.cpp

8.25 Square Class Reference

Class for squares that makes grid of level.

```
#include <square.hpp>
```

Public Member Functions

- Square (Vector2D center)
  Construct a new Square object.
- ~**Square** ()
  Destroy the Square object.
- Vector2D get_center () const
  Returns center cordinates of square.
- int get_occupied () const
  Returns what is occupying square.
- void **print_info** ()
  Prints info of square What is squares center point and by what square is occupied.
- bool occupy_by_grass ()
  Function to occupy square by grass.
- bool occupy_by_road ()
  Function to occupy square by road.
- bool occupy_by_tower ()
  Function to occupy square by tower.

8.25.1 Detailed Description

Class for squares that makes grid of level.

Parameters

center	cordinates of squares center point in Vector2D

8.25.2 Constructor & Destructor Documentation

8.25.2.1 Square()

```
Square::Square (
            Vector2D center )
```

Construct a new Square object.

Parameters

center	cordinates in Vector2D where are square center

8.25.3 Member Function Documentation

8.25.3.1 get_center()

```
Vector2D Square::get_center ( ) const
```

Returns center cordinates of square.

Returns

Vector2D

8.25.3.2 get_occupied()

```
int Square::get_occupied ( ) const
```

Returns what is occupying square.

Returns

int = enum occupied_by

8.25.3.3 occupy_by_grass()

```
bool Square::occupy_by_grass ( )
```

Function to occupy square by grass.

Returns

true if was successful

false if wasn't

8.25.3.4 occupy_by_road()

```
bool Square::occupy_by_road ( )
```

Function to occupy square by road.

Returns

true if was successful

false if wasn't

8.25.3.5 occupy_by_tower()

```
bool Square::occupy_by_tower ( )
```

Function to occupy square by tower.

Returns

true if was successful

false if wasn't

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

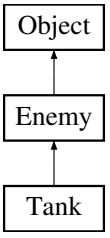
- src/square.hpp
- src/square.cpp

8.26 Tank Class Reference

Class for a Tank enemy.

```
#include <tank_enemy.hpp>
```

Inheritance diagram for Tank:



Public Member Functions

- Tank (Level &level, Vector2D &position, int health=700, int damage=20, int range=130, int attack_speed=20, int type=ObjectTypes::TankOrc, int speed=2, int defense=30, int size=4)

  Constructor to initialize a Tank Enemy object.
- ~**Tank** ()

  Destructor for the Tank Enemy.
- bool attack ()

  Perform an attack action specific to the Tank Enemy.

Public Member Functions inherited from Enemy

- Enemy (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int speed, int defense, int size)

  Constructor to initialize an Enemy object.
- ~**Enemy** ()

  Destructor for the Enemy.
- int get_speed () const

  Get the speed attribute of the Enemy.
- int get_original_speed () const

  Get the original speed attribute of the Enemy.
- int get_defense () const

  Get the defense attribute of the Enemy.
- int get_size () const

  Get the size of the Enemy.
- void lose_speed (int amount)

  Decrease the speed attribute of the Enemy.
- void set_speed (int amount)

  Set the speed attribute of the Enemy.
- void **move** ()

  Move the Enemy based on its behavior logic.
- std::vector< Vector2D > get_route () const

  Get the route of the Enemy.
- void set_route_position (Vector2D position)

  Set the route position for the Enemy.
- Vector2D get_prev_pos ()

  Get the previous position of the Enemy.
- void set_prev_pos (Vector2D pos)

  Set the previous position of the Enemy.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.

- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.

8.26.1 Detailed Description

Class for a Tank enemy.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 Tank()

```
Tank::Tank (
            Level & level,
            Vector2D & position,
            int health = 700,
            int damage = 20,
            int range = 130,
            int attack_speed = 20,
            int type = ObjectTypes::TankOrc,
            int speed = 2,
            int defense = 30,
            int size = 4 )
```

Constructor to initialize a Tank Enemy object.

Parameters

level	Reference to the Level object.
position	Initial position of the Tank Enemy (Vector2D).
health	Initial health points of the Tank Enemy.
damage	Damage inflicted by the Tank Enemy.
range	Range of attack for the Tank Enemy.
attack_speed	Speed of attack for the Tank Enemy.
type	Type of the Tank Enemy.
speed	Speed attribute of the Tank Enemy.
defense	Defense attribute of the Tank Enemy.
size	Size of the Tank Enemy.

8.26.3 Member Function Documentation

8.26.3.1 attack()

```
bool Tank::attack ( )  [virtual]
```

Perform an attack action specific to the Tank Enemy.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

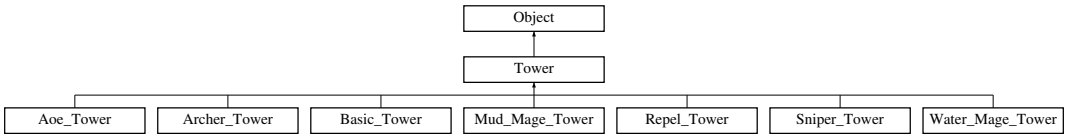
- src/tank_enemy.hpp
- src/tank_enemy.cpp

8.27 Tower Class Reference

Tower class for handling all towers.

`#include <tower.hpp>`

Inheritance diagram for Tower:



Public Member Functions

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.

- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

  Decrease the health value of the object.
- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.
- State get_state ()

  Get the current state of the object.
- int get_wait_time () const

  Get the reset wait time for all objects.
- void set_state (State state)

  Set the state of the object.
- virtual bool attack ()

  Virtual method for Object's attack action.

8.27.1 Detailed Description

Tower class for handling all towers.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 Tower()

```
Tower::Tower (
            Level & current_level,
            Vector2D & position,
            int health,
            int damage,
            int range,
            int attack_speed,
            int type,
            int price,
            int level )
```

Constructor to initialize a Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Tower (Vector2D).
health	Initial health points of the Tower.
damage	Damage inflicted by the Tower.
range	Range of attack for the Tower.
attack_speed	Speed of attack for the Tower.
type	Type of the Tower.
price	Price of the Tower.
level	Level of the Tower.

8.27.3 Member Function Documentation

8.27.3.1 get_level()

```
int Tower::get_level ( )
```

Get the level of the Tower, that is how many times the tower has been upgraded.

Returns

Integer representing the level of the Tower.

8.27.3.2 get_price()

```
int Tower::get_price ( )
```

Get the price of the Tower.

Returns

Integer representing the price of the Tower.

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

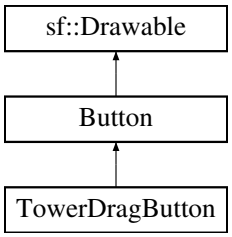
- src/tower.hpp
- src/tower.cpp

8.28 TowerDragButton Class Reference

A class to implement dragging towers to grid.

`#include <tower_drag_button.hpp>`

Inheritance diagram for TowerDragButton:



Public Member Functions

- **TowerDragButton** (int type, sf::Vector2f position, sf::Color outline, sf::Color fill, ResourceHandler &rh)
- void **set_drag_flag** ()

  Set the drag flag to true.
- void **reset_drag_flag** ()

  Set drag flag to false.
- bool get_drag_flag () const

  Get the drag flag object.
- const sf::Sprite * get_dragging_image () const

  Get the dragging image sf::Sprite. Used in SideMenu to draw the dragging image on top of everything.
- int get_type () const

  Get the ObjectType of the button.
- void update (int player_cash)

  Updates the buttons state. If cash is too low for purchase, sets the price text to red and disables the events.
- void handle_events (sf::RenderWindow &window, const sf::Event &event, Level &lv)

  Handles the button events. If mouse is pressed on top of the button sets the drag flag on. If mouse moves sets the new dragging image position. If mouse button is released, calls add_tower_to_release_square() and resets drag flag.

Public Member Functions inherited from Button

- **Button** (const std::string &label, sf::Vector2f size, sf::Vector2f position, sf::Color fill_color, sf::Color outline_color, const sf::Font &font)
- void **center_text** ()

  Centers the sf::text object to the middle of the button.
- void set_outline_color (sf::Color outline_color)

  Sets outline color of the button box and color of the text.
- void set_fill_color (sf::Color fill_color)

  Sets fill color of the button box.
- void set_text_string (const std::string &label)

  Sets the button's text as the given string.
- void set_position (sf::Vector2f pos)

  Sets the position of the button background and centers the text according the button and text size.
- void set_size (sf::Vector2f size)

Sets the size of the button.

- void set_font (const sf::Font &font)

  assigns the given reference to a font to the buttons sf::Text object.
- sf::Vector2f get_size ()

  Returns the buttons size.
- bool button_pressed ()

  Returns a true value if the button is pressed. Used in if-statements to trigger actions in menus.
- void reset_button ()

  Resets the button. Used after button_pressed in menus.
- bool is_mouse_over (sf::RenderWindow &window)

  Resets the button. Used after button_pressed in menus.
- void handle_events (sf::RenderWindow &window, const sf::Event &event, Level &lv)

  Handle mouse events given from the game class. Makes fill color brighter if mouse hovers on top of button. Makes button less transparent if mouse hovers on top of button (used as transparent in MainMenu and ChooseLevelMenu). Sets button pressed if mouse click is recorded on top of button.

Protected Member Functions

- void add_tower_to_release_square (sf::RenderWindow &window, Level &lv)

  adds tower to the release square. Checks if the object is on the grid, if the square is grass, and handles the purchase with the level class if add_tower function succeeds.
- void set_dragging_drawable_pos (sf::RenderWindow &window)

  Set the dragging drawable position. Checks the mouse position and sets it to the member variable _release_pos.
- void **setup_tower_images** ()

  Sets up the tower image sprites to right locations, sizes, etc. Called in constructor.
- void **setup_button_texts** ()

  Sets up the button sf::Texts objects to right locations etc.
- void **setup_attribute_images** ()

  Sets up the attribute images to close to the texts.
- void setup_attribute_image (int type, sf::Sprite &sprite, sf::Vector2f pos)

  a helper function for the function above Sets up a sprite according to given position and ObjectType.
- void **setup_font** ()

  Sets up the font to all texts.
- void draw (sf::RenderTarget &target, sf::RenderStates states) const

  Inherited from the sf::Drawable class, so that the button can be drawn as window.draw(Button). Draws the static tower image of object, bounding square, object attribute texts.

Protected Attributes

- sf::RectangleShape **_img_background**

  a background for tower texture
- sf::Sprite **_drawable_tower**

  a sprite for the static tower image
- sf::Sprite **_drawable_dragging_tower**

  a sprite for the tower image following the mouse when dragging
- sf::Sprite **_hp_img**

  a sprite for the static tower image
- sf::Sprite **_dmg_img**

  a sprite for displaying damage image
- sf::Sprite **_rng_img**

a sprite for displaying the range image

- sf::Sprite **_atkspd_img**

  a sprite for displaying the attack speed image
- sf::Text **_price_text**

  a text object for displaying the towers price
- sf::Text **_hp_text**

  a text object for displaying the hp of the tower
- sf::Text **_dmg_text**

  a text object for displaying the damage of the tower
- sf::Text **_rng_text**

  a text object for displaying the range of the tower
- sf::Text **_atkspd_text**

  a text object for displaying the attack speed of the tower
- sf::Vector2i **_release_pos**

  the position where the mouse was released when drag flag is set
- float **_img_size** = 60

  size of the tower images.
- float **_attr_img_size** = 12

  size of the attribute images (hp, dmg, etc.)
- int **_tower_type**

  tower's ObjectType
- int **_tower_price**

  tower's price, displayed in _price_text
- bool **_button_enabled**

  Determines if the button is usable.
- bool **_drag_flag**

  Determines if the image is dragged or not.
- ResourceHandler & **_rh**

  Handles Tower textures, font, name, attribute values and images from given tower type.

Protected Attributes inherited from Button

- sf::RectangleShape **_button**

  Background of the button.
- sf::Color **_button_fill_color**

  Fill color of the button.
- sf::Color **_button_outline_color**

  Outline color of the button.
- sf::Vector2f **_position**

  Position of the button.
- sf::Vector2f **_size**

  Size of the button.
- sf::Text **_text**

  Text obect of the button.
- bool **_button_pressed**

  The state of the button.

Additional Inherited Members

Static Public Member Functions inherited from Button

- static bool inside_grid (sf::Vector2i mouse_pos, Level &lv)

  Checks if mouse position is inside grid.
- static std::pair< int, int > window_coords_to_grid_index (sf::Vector2i mouse_pos, Level &lv)

  calculates and returns grid index (column, row) of the level from the mouse position w.r.t window
- static Vector2D window_coords_to_level_coords (sf::Vector2i mouse_pos)

  calculates and returns Vector2D coords of the level from the mouse position w.r.t window

8.28.1 Detailed Description

A class to implement dragging towers to grid.

Parameters

type	what object the button creates
position	location for the button
outline	outline color of the button
fill	fill color of the button
rh	a reference to ResourceHandler class

8.28.2 Member Function Documentation

8.28.2.1 add_tower_to_release_square()

```
void TowerDragButton::add_tower_to_release_square (
            sf::RenderWindow & window,
            Level & lv )  [protected]
```

adds tower to the release square. Checks if the object is on the grid, if the square is grass, and handles the purchase with the level class if add_tower function succeeds.

Parameters

window	
lv	

8.28.2.2 draw()

```
void TowerDragButton::draw (
            sf::RenderTarget & target,
            sf::RenderStates states ) const  [protected], [virtual]
```

Inherited from the sf::Drawable class, so that the button can be drawn as window.draw(Button). Draws the static tower image of object, bounding square, object attribute texts.

Parameters

target	
states	

Reimplemented from Button.

8.28.2.3 get_drag_flag()

```
bool TowerDragButton::get_drag_flag ( ) const
```

Get the drag flag object.

Returns

true

false

8.28.2.4 get_dragging_image()

```
const sf::Sprite * TowerDragButton::get_dragging_image ( ) const
```

Get the dragging image sf::Sprite. Used in SideMenu to draw the dragging image on top of everything.

Returns

const sf::Sprite*

8.28.2.5 get_type()

```
int TowerDragButton::get_type ( ) const
```

Get the ObjectType of the button.

Returns

int

8.28.2.6 handle_events()

```
void TowerDragButton::handle_events (
            sf::RenderWindow & window,
            const sf::Event & event,
            Level & lv )
```

Handles the button events. If mouse is pressed on top of the button sets the drag flag on. If mouse moves sets the new dragging image position. If mouse button is released, calls add_tower_to_release_square() and resets drag flag.

Parameters

window	
event	
lv	

8.28.2.7 set_dragging_drawable_pos()

```
void TowerDragButton::set_dragging_drawable_pos (
            sf::RenderWindow & window )  [protected]
```

Set the dragging drawable position. Checks the mouse position and sets it to the member variable _release_pos.

Parameters

window	

8.28.2.8 setup_attribute_image()

```
void TowerDragButton::setup_attribute_image (
            int type,
            sf::Sprite & sprite,
            sf::Vector2f pos )  [protected]
```

a helper function for the function above Sets up a sprite according to given position and ObjectType.

Parameters

type	
sprite	
pos	

8.28.2.9 update()

```
void TowerDragButton::update (
            int player_cash )
```

Updates the buttons state. If cash is too low for purchase, sets the price text to red and disables the events.

Parameters

player_cash	

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

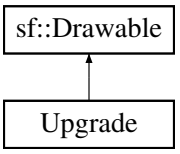
- src/tower_drag_button.hpp
- src/tower_drag_button.cpp

8.29 Upgrade Class Reference

A class to implement tower upgrades. Click a object on the grid and a menu will pop. Push upgrade button to upgrade the tower's attributes. Basic idea:

`#include <upgrade.hpp>`

Inheritance diagram for Upgrade:



Public Member Functions

- **Upgrade** (float grid_resolution, ResourceHandler &rh, Level &level, int upgrade_cost, int max_upgrades)
- **Upgrade** (const Upgrade &)=delete
- Upgrade & **operator=** (const Upgrade &)=delete
- void handle_events (sf::RenderWindow &window, sf::Event &event)

  Handles events from the game.
- void **disable_grid_click** ()

  Disable grid click.
- void **reset** ()

  Reset upgrade so the menu wont stay to next game.

8.29.1 Detailed Description

A class to implement tower upgrades. Click a object on the grid and a menu will pop. Push upgrade button to upgrade the tower's attributes. Basic idea:

1. click grid position.
2. if there is a tower: pop menu with current attributes of the tower on the square
3. press button to call tower_to_be_upgraded->level_up()
4. hide menu

Parameters

grid_resolution	
rh	
level	
upgrade_cost	

8.29.2 Member Function Documentation

8.29.2.1 handle_events()

```
void Upgrade::handle_events (
            sf::RenderWindow & window,
            sf::Event & event )
```

Handles events from the game.

Parameters

window	
event	

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

- src/upgrade.hpp
- src/upgrade.cpp

8.30 Vector2D Class Reference

class for in game coordinates

```
#include <vector2d.hpp>
```

Public Member Functions

- Vector2D (int x, int y)
  Construct a new Vector 2 D object.
- bool operator== (const Vector2D &other) const
  Overriding ==.
- bool operator!= (const Vector2D &other) const
  Overriding !=.
- Vector2D operator+ (const Vector2D &other) const
  Overriding addition operator (+) for Vector2D objects.
- Vector2D operator- (const Vector2D &other) const
  Overriding subtraction operator (-) for Vector2D objects.

Public Attributes

- int **x**
- int **y**

Friends

- std::ostream & operator<< (std::ostream &os, const Vector2D &vec)
  Overriding <<.

8.30.1 Detailed Description

class for in game coordinates

8.30.2 Constructor & Destructor Documentation

8.30.2.1 Vector2D()

```
Vector2D::Vector2D (
            int x,
            int y )  [inline]
```

Construct a new Vector 2 D object.

Parameters

x	
y	

8.30.3 Member Function Documentation

8.30.3.1 operator"!=()

```
bool Vector2D::operator!= (
            const Vector2D & other ) const  [inline]
```

Overriding !=.

Parameters

other	other Vector2D that current isn't matched with

Returns

true if two Vector2D aren't identical

false if they are identical

8.30.3.2 operator+()

```
Vector2D Vector2D::operator+ (
            const Vector2D & other ) const  [inline]
```

Overriding addition operator (+) for Vector2D objects.

Parameters

other	Vector2D object to be added to the current Vector2D.

Returns

Vector2D Resultant Vector2D object after addition.

8.30.3.3 operator-()

```
Vector2D Vector2D::operator- (
            const Vector2D & other ) const  [inline]
```

Overriding subtraction operator (-) for Vector2D objects.

Parameters

other	Vector2D object to be subtracted from the current Vector2D.

Returns

Vector2D Resultant Vector2D object after subtraction.

8.30.3.4 operator==()

```
bool Vector2D::operator== (
            const Vector2D & other ) const  [inline]
```

Overriding ==.

Parameters

other	other Vector2D that current is matched with

Returns

true if two Vector2D are identical

false if they are not

8.30.4 Friends And Related Symbol Documentation

8.30.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Vector2D & vec )  [friend]
```

Overriding <<.

Parameters

os	where output is given
vec	Vector2D that information is beeing printed

Returns

std::ostream& new line in console with information about this vector in format "x y"

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

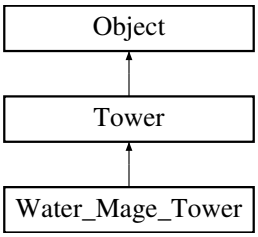
- src/vector2d.hpp

8.31 Water_Mage_Tower Class Reference

Class for a Water Mage tower.

`#include <water_mage_tower.hpp>`

Inheritance diagram for Water_Mage_Tower:



Public Member Functions

- Water_Mage_Tower (Level &current_level, Vector2D &position, int health=300, int damage=40, int range=150, int attack_speed=25, int type=ObjectTypes::WaterMageTower, int price=250, int level=1)

  Constructor to initialize a Water Mage Tower object.
- ~**Water_Mage_Tower** ()

  Destructor for the Water Mage Tower.
- bool attack ()

  Perform an attack action specific to the Water Mage Tower.

Public Member Functions inherited from Tower

- Tower (Level &current_level, Vector2D &position, int health, int damage, int range, int attack_speed, int type, int price, int level)

  Constructor to initialize a Tower object.
- ~**Tower** ()

  Destructor for the Tower.
- void **level_up** ()

  Increase the level of the Tower.
- int get_price ()

  Get the price of the Tower.
- int get_level ()

  Get the level of the Tower, that is how many times the tower has been upgraded.

Public Member Functions inherited from Object

- Object (Level &level, Vector2D &position, int health, int damage, int range, int attack_speed, int type)

  Constructor to initialize an Object.
- virtual ~**Object** ()

  Virtual destructor for Object.
- int get_damage () const

  Get the damage value of the object.
- int get_health () const

  Get the health value of the object.
- int get_range () const

  Get the attack range of the object.
- int get_attack_speed () const

  Get the attack speed of the object.
- int get_original_attack_speed () const

  Get the original attack speed of the object.
- const Vector2D get_position () const

  Get the position of the object.
- int get_type () const

  Get the type of the object.
- Level & get_level_reference () const

  Get a reference to the Level object associated with this object.
- int get_attack_counter () const

  Get the attack counter value of the object.
- int get_reset_counter () const

  Get the reset counter value of the object.
- void set_position (const Vector2D &position)

  Set the position of the object.
- void set_attack_counter (const int amount)

  Set the attack counter value of the object.
- void set_reset_counter (const int amount)

  Set the reset counter value of the object.
- void set_attack_speed (const int amount)

  Set the attack speed of the object.
- void set_original_attack_speed (const int amount)

  Set the original attack speed of the object.
- void **attack_counter_up** ()

  Increment the attack counter value by one.
- void **reset_counter_up** ()

  Increment the reset counter value by one.
- void gain_damage (int amount)

  Increase the damage value of the object.
- void gain_health (int amount)

  Increase the health value of the object.
- void gain_range (int amount)

  Increase the attack range of the object.
- void gain_attack_speed (int amount)

  Increase the attack speed of the object.
- double distance_to (const Vector2D &target_position)

  Calculate the distance between the object and a target position.
- void lose_health (int amount)

Decrease the health value of the object.

- void lose_attack_speed (int amount)

  Decrease the attack speed of the object.

- State get_state ()

  Get the current state of the object.

- int get_wait_time () const

  Get the reset wait time for all objects.

- void set_state (State state)

  Set the state of the object.

8.31.1 Detailed Description

Class for a Water Mage tower.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 Water_Mage_Tower()

```
Water_Mage_Tower::Water_Mage_Tower (
            Level & current_level,
            Vector2D & position,
            int health = 300,
            int damage = 40,
            int range = 150,
            int attack_speed = 25,
            int type = ObjectTypes::WaterMageTower,
            int price = 250,
            int level = 1 )
```

Constructor to initialize a Water Mage Tower object.

Parameters

current_level	Reference to the Level object.
position	Initial position of the Water Mage Tower (Vector2D).
health	Initial health points of the Water Mage Tower.
damage	Damage inflicted by the Water Mage Tower.
range	Range of attack for the Water Mage Tower.
attack_speed	Speed of attack for the Water Mage Tower.
type	Type of the Water Mage Tower.
price	Price of the Water Mage Tower.
level	Level of the Water Mage Tower.

8.31.3 Member Function Documentation

8.31.3.1 attack()

```
bool Water_Mage_Tower::attack ( )  [virtual]
```

Perform an attack action specific to the Water Mage Tower.

Returns

true if the attack is successful, false otherwise.

Reimplemented from Object.

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

- src/water_mage_tower.hpp
- src/water_mage_tower.cpp

Chapter 9

File Documentation

9.1 aoe_tower.hpp

```
00001 #ifndef AOE_TOWER_HPP
00002 #define AOE_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Aoe_Tower : public Tower {
00012 public:
00026     Aoe_Tower(Level& current_level, Vector2D& position, int health = 400, int damage = 50, int range =
      200,
00027               int attack_speed = 25, int type = ObjectTypes::AoeTower, int price = 150, int level =
      1);
00028
00032     ~Aoe_Tower() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.2 archer_tower.hpp

```
00001 #ifndef ARCHER_TOWER_HPP
00002 #define ARCHER_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Archer_Tower : public Tower {
00012 public:
00026     Archer_Tower(Level& current_level, Vector2D& position, int health = 300, int damage = 25, int
      range = 250,
00027                  int attack_speed = 20, int type = ObjectTypes::ArcherTower, int price = 100, int
      level = 1);
00028
00032     ~Archer_Tower() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.3 attack_types.hpp

```
00001 #ifndef ATTACK_TYPES_HPP
00002 #define ATTACK_TYPES_HPP
00003
```

```
00004 #include <string>
00005 #include <iostream>
00006
00007 class Tower;
00008 class Enemy;
00009
00013 namespace ObjectTypes {
00017     enum Enemies {
00018         NoobSkeleton_NoAttack,
00019         NoobDemon_CanAttack,
00020         FastBoy,
00021         FogMage,
00022         HealerPriest,
00023         InfernoMage,
00024         TankOrc,
00025         BossKnight,
00026     };
00027
00031     enum Towers {
00032         AoeTower,
00033         ArcherTower,
00034         MudMageTower,
00035         RepelMageTower,
00036         SniperTower,
00037         WaterMageTower,
00038     };
00039 }
00040
00048 double check_type_multiplier(Tower* tower, Enemy* enemy);
00049
00050 #endif
```

9.4 basic_enemy.cpp

```
00001 #include "basic_enemy.hpp"
00002 #include "level.hpp"
00003
00004 Basic_Enemy::Basic_Enemy(Level& level, Vector2D& position, int health, int damage, int range, int
      attack_speed, int type, int speed, int defense, int size) :
00005     Enemy(level, position, health, damage, range, attack_speed, type, speed, defense, size) {}
00006
00007 bool Basic_Enemy::attack() {
00008     double multiplier;
00009     Level& level_reference = get_level_reference();
00010
00011     if (!level_reference.get_towers().empty()) {
00012         for (auto& tower : level_reference.get_towers()) {
00013
00014             double dist = this->distance_to(tower->get_position());
00015
00016             if (dist <= this->get_range()) {
00017                 multiplier = check_type_multiplier(tower, this);
00018                 tower->lose_health(this->get_damage() * multiplier);
00019                 if (this->get_position().y > tower->get_position().y){
00020                     set_state(State::attacking_left);
00021                 } else {
00022                     set_state(State::attacking_right);
00023                 }
00024                 return true;
00025                 break;
00026             }
00027         }
00028         std::this_thread::sleep_for(std::chrono::milliseconds(this->get_attack_speed()));
00029     }
00030     set_state(State::none);
00031     return false;
00032 }
```

9.5 basic_enemy.hpp

```
00001 #ifndef BASIC_ENEMY_HPP
00002 #define BASIC_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006 // #include "level.hpp"
00007
00011 class Basic_Enemy : public Enemy {
00012 public:
```

```
00027     Basic_Enemy(Level& level, Vector2D& position, int health = 20, int damage = 5, int range = 100,
      int attack_speed = 1,
00028                 int type = ObjectTypes::NoobDemon_CanAttack, int speed = 20, int defense = 5, int size
      = 3);
00029
00033     ~Basic_Enemy() { }
00034
00040     bool attack();
00041 };
00042
00043 #endif
```

9.6 basic_tower.cpp

```
00001 #include "basic_tower.hpp"
00002
00003 Basic_Tower::Basic_Tower(Level& current_level, Vector2D& position, int health, int damage, int range,
00004         int attack_speed, int type, int price, int level, bool attack_type_single) :
00005
00006         Tower(current_level, position, health, damage, range, attack_speed, type, price, level),
      _attack_type_single(attack_type_single) {}
00007
00008 bool Basic_Tower::attack() {
00009     double multiplier;
00010     Level& level_reference = get_level_reference();
00011
00012     if (!level_reference.get_enemies().empty()) {
00013         if (_attack_type_single) {
00014             for (auto& enemy : level_reference.get_enemies()) {
00015
00016                 double dist = this->distance_to(enemy->get_position());
00017
00018                 if (dist <= this->get_range()) {
00019                     multiplier = check_type_multiplier(this, enemy);
00020                     enemy->lose_health(this->get_damage() * multiplier);
00021                     if (this->get_position().y > enemy->get_position().y){
00022                         set_state(State::attacking_left);
00023                     } else {
00024                         set_state(State::attacking_right);
00025                     }
00026                     return true;
00027                     break;
00028                 }
00029             }
00030         } else {
00031             int counter = 0;
00032             if (this->get_position().y > level_reference.get_enemies()[0]->get_position().y){
00033                 set_state(State::attacking_left);
00034             } else {
00035                 set_state(State::attacking_right);
00036             }
00037             for (auto& enemy : level_reference.get_enemies()) {
00038
00039                 double dist = this->distance_to(enemy->get_position());
00040
00041                 if (dist <= this->get_range()) {
00042                     multiplier = check_type_multiplier(this, enemy);
00043                     enemy->lose_health(this->get_damage() * multiplier);
00044                     counter++;
00045                 }
00046
00047                 if (counter >= 3) {
00048                     return true;
00049                     break;
00050                 }
00051             }
00052         }
00053         std::this_thread::sleep_for(std::chrono::milliseconds(this->get_attack_speed()));
00054     }
00055     set_state(State::none);
00056     return false;
00057 }
00058
00059 void Basic_Tower::set_multiple_target() {
00060     _attack_type_single = false;
00061 }
```

9.7 basic_tower.hpp

```
00001 #ifndef BASIC_TOWER_HPP
```

```
00002 #define BASIC_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Basic_Tower : public Tower {
00012 public:
00027     Basic_Tower(Level& current_level, Vector2D& position, int health = 30, int damage = 10, int range
      = 100,
00028                 int attack_speed = 1, int type = ObjectTypes::ArcherTower, int price = 100, int level
      = 1, bool attack_type_single = true);
00029
00033     ~Basic_Tower() { }
00034
00040     bool attack();
00041
00045     void set_multiple_target();
00046
00047 private:
00048     bool _attack_type_single;
00049 };
00050
00051 #endif
```

9.8 boss_enemy.hpp

```
00001 #ifndef BOSS_ENEMY_HPP
00002 #define BOSS_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Boss : public Enemy {
00011 public:
00026     Boss(Level& level, Vector2D& position, int health = 800, int damage = 20, int range = 150, int
      attack_speed = 40,
00027          int type = ObjectTypes::BossKnight, int speed = 1, int defense = 20, int size = 5);
00028
00032     ~Boss() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.9 button.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_BUTTON
00002 #define TOWER_DEFENCE_SRC_BUTTON
00003
00004 #include <SFML/Window.hpp>
00005 #include <SFML/Graphics.hpp>
00006 #include "level.hpp"
00007 #include "vector2d.hpp"
00008 #include <iostream>
00009
00021 class Button : public sf::Drawable{
00022
00023 public:
00024
00025     Button(const std::string& label, sf::Vector2f size, sf::Vector2f position, sf::Color fill_color,
      sf::Color outline_color, const sf::Font& font);
00026
00030     void center_text();
00031
00036     void set_outline_color(sf::Color outline_color);
00037
00042     void set_fill_color(sf::Color fill_color);
00043
00048     void set_text_string(const std::string& label);
00049
00054     void set_position(sf::Vector2f pos);
00055
00060     void set_size(sf::Vector2f size);
00061
00066     void set_font(const sf::Font& font);
00067
00068
```

```
00073     sf::Vector2f get_size();
00074
00079     bool button_pressed();
00080
00085     void reset_button();
00086
00091     bool is_mouse_over(sf::RenderWindow& window);
00092
00104     void handle_events(sf::RenderWindow& window, const sf::Event& event, Level& lv);
00105
00113     static bool inside_grid( sf::Vector2i mouse_pos, Level& lv);
00114
00115
00123     static std::pair<int, int> window_coords_to_grid_index( sf::Vector2i mouse_pos, Level& lv);
00124
00132     static Vector2D window_coords_to_level_coords(sf::Vector2i mouse_pos);
00133
00134
00135
00136 protected:
00137
00143     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
00144
00148     sf::RectangleShape _button;
00149
00153     sf::Color _button_fill_color;
00154
00158     sf::Color _button_outline_color;
00159
00163     sf::Vector2f _position;
00164
00168     sf::Vector2f _size;
00169
00174     sf::Text _text;
00175
00179     bool _button_pressed;
00180
00181 };
00182
00183
00184
00185
00186
00187
00188 #endif
```

9.10 choose_level_menu.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_CHOOSE_LEVEL_MENU
00002 #define TOWER_DEFENCE_SRC_CHOOSE_LEVEL_MENU
00003
00004
00005 #include <SFML/Window.hpp>
00006 #include <SFML/Graphics.hpp>
00007 #include "level.hpp"
00008 #include "resource_handler.hpp"
00009 #include "button.hpp"
00010
00011 #include <iostream>
00012
00019 class ChooseLevelMenu : public sf::Drawable{
00020 public:
00021
00022     ChooseLevelMenu( ResourceHandler& rh, Level& level);
00023     ~ChooseLevelMenu(){
00024     delete _level1_button;
00025     delete _level2_button;
00026     delete _level3_button;
00027     delete _level4_button;
00028     delete _level5_button;
00029     delete _level6_button;
00030     }
00031
00032     ChooseLevelMenu(const ChooseLevelMenu&) = delete;
00033     ChooseLevelMenu& operator=(const ChooseLevelMenu&) = delete;
00034
00035
00040     void disable_menu();
00041
00046     void enable_menu();
00047
00055     int get_state();
00056
```

```
00061     void reset();
00062
00069     void handle_events(sf::RenderWindow& window, sf::Event& event);
00070
00076     const std::string get_level_to_load();
00077
00078
00079 private:
00080
00085     void set_menu_background();
00086
00091     void set_buttons();
00092
00099     void draw(sf::RenderTarget& target, sf::RenderStates state) const;
00100
00105     bool _menu_enabled;
00106
00113     int _level_number;
00114
00119     int _state;
00120
00125     sf::Sprite _menu_background;
00126
00127
00132     Button* _level1_button;
00133
00138     Button* _level2_button;
00139
00144     Button* _level3_button;
00145
00150     Button* _level4_button;
00151
00156     Button* _level5_button;
00157
00162     Button* _level6_button;
00163
00168     ResourceHandler& _rh;
00169
00174     Level& _level;
00175
00176
00177 };
00178
00179
00180 #endif
```

9.11 demon_enemy.hpp

```
00001 #ifndef DEMON_ENEMY_HPP
00002 #define DEMON_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Demon : public Enemy {
00011 public:
00026     Demon(Level& level, Vector2D& position, int health = 150, int damage = 10, int range = 130, int
      attack_speed = 20,
00027           int type = ObjectTypes::NoobDemon_CanAttack, int speed = 2, int defense = 5, int size = 2);
00028
00032     ~Demon() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.12 enemy.hpp

```
00001 #ifndef ENEMY_HPP
00002 #define ENEMY_HPP
00003
00004 #include "object.hpp"
00005
00009 class Enemy : public Object {
00010 public:
00025     Enemy(Level& level, Vector2D& position, int health, int damage, int range, int attack_speed, int
      type, int speed, int defense, int size);
00026
```

```
00030     ~Enemy() { }
00031
00037     int get_speed() const;
00038
00044     int get_original_speed() const;
00045
00051     int get_defense() const;
00052
00058     int get_size() const;
00059
00065     void lose_speed(int amount);
00066
00072     void set_speed(int amount);
00073
00077     void move();
00078
00079     // void State get_state(); // Uncomment and add Doxygen-style comment if required
00080
00081     // bool attack(); // Uncomment and add Doxygen-style comment if required
00082
00088     std::vector<Vector2D> get_route() const;
00089
00095     void set_route_position(Vector2D position);
00096
00102     Vector2D get_prev_pos();
00103
00109     void set_prev_pos(Vector2D pos);
00110
00111 private:
00112     Vector2D _prev_pos;
00113     int _speed;
00114     int _original_speed = 0;
00115     int _defense;
00116     int _size;
00117     std::vector<Vector2D> _route;
00118 };
00119
00120 #endif
```

9.13 fastboy_enemy.hpp

```
00001 #ifndef FASTBOY_ENEMY_HPP
00002 #define FASTBOY_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Fast_Boy : public Enemy {
00011 public:
00026     Fast_Boy(Level& level, Vector2D& position, int health = 10, int damage = 0, int range = 0, int
      attack_speed = 1,
00027              int type = ObjectTypes::FastBoy, int speed = 10, int defense = 5, int size = 2);
00028
00032     ~Fast_Boy() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.14 fogmage_enemy.hpp

```
00001 #ifndef FOGMAGE_ENEMY_HPP
00002 #define FOGMAGE_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Fog_Mage : public Enemy {
00011 public:
00026     Fog_Mage(Level& level, Vector2D& position, int health = 300, int damage = 0, int range = 150, int
      attack_speed = 50,
00027              int type = ObjectTypes::FogMage, int speed = 1, int defense = 10, int size = 4);
00028
00032     ~Fog_Mage() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.15 game.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_GAME
00002 #define TOWER_DEFENCE_SRC_GAME
00003
00004 #include "level.hpp"
00005 #include <SFML/Window.hpp>
00006 #include <SFML/Graphics.hpp>
00007 #include <iostream>
00008 #include "renderer.hpp"
00009 #include <random>
00010
00011 #include "basic_enemy.hpp"
00012 #include "basic_tower.hpp"
00013 #include "aoe_tower.hpp"
00014 #include "archer_tower.hpp"
00015 #include "boss_enemy.hpp"
00016 #include "demon_enemy.hpp"
00017 #include "fastboy_enemy.hpp"
00018 #include "fogmage_enemy.hpp"
00019 #include "healer_enemy.hpp"
00020 #include "inferno_enemy.hpp"
00021 #include "repel_tower.hpp"
00022 #include "sceleton_enemy.hpp"
00023 #include "sniper_tower.hpp"
00024 #include "tank_enemy.hpp"
00025
00026 #include "side_menu.hpp"
00027 #include "upgrade.hpp"
00028 #include "main_menu.hpp"
00029 #include "choose_level_menu.hpp"
00030
00031
00032
00037 namespace GameState{
00038     enum State{
00039         StartMenu, MapMenu, Pause, Round, Victory, GameOver
00040     };
00041 }
00042
00043
00047 class Game{
00048 public:
00052     Game();
00056     ~Game(){}
00057     Game(const Game& ) = delete;
00058     Game operator=(const Game&) = delete;
00059
00064     int get_side_bar_width() const;
00069     int get_game_resolution() const;
00070
00074     void run();
00075
00076 private:
00077
00081     void open_window();
00082
00086     void process_events();
00087
00091     void update();
00092
00096     void render();
00097
00101     void start_round();
00102
00107     void update_enemies();
00108
00113     void update_towers();
00114
00115     int _game_resolution;
00116     int _side_bar_width;
00117
00118     sf::RenderWindow _window;
00119
00120
00121
00126     sf::Time _reset_time = sf::Time::Zero;
00127
00132     sf::Clock _reset_clock;
00133
00134
00139     int _starting_cash = 500;
00140
00144     int _starting_lives = 30;
00145
00150     int _game_state = 0;
00151
```

```
00155     int _rounds_to_survive = 30;
00156
00160     bool _round_over = true;
00161
00165     int _difficulty_multiplier = 2;
00166
00171     int _available_types = 1;
00172
00178     int _enemy_move_animation;
00179
00183     int _basic_money = 50;
00184
00189     ResourceHandler _rh;
00190
00195     Renderer _renderer;
00196
00201     Level _level;
00202
00207     MainMenu _main_menu;
00208
00213     ChooseLevelMenu _level_menu;
00214
00219     SideMenu _side_menu;
00220
00225     Upgrade _upgrade;
00226 };
00227 #endif
```

9.16 healer_enemy.hpp

```
00001 #ifndef HEALER_ENEMY_HPP
00002 #define HEALER_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Healer : public Enemy {
00011 public:
00026     Healer(Level& level, Vector2D& position, int health = 200, int damage = 0, int range = 100, int
      attack_speed = 20,
00027            int type = ObjectTypes::HealerPriest, int speed = 1, int defense = 10, int size = 2);
00028
00032     ~Healer() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.17 inferno_enemy.hpp

```
00001 #ifndef INFERNO_ENEMY_HPP
00002 #define INFERNO_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Inferno : public Enemy {
00011 public:
00026     Inferno(Level& level, Vector2D& position, int health = 400, int damage = 40, int range = 150, int
      attack_speed = 60,
00027             int type = ObjectTypes::InfernoMage, int speed = 3, int defense = 10, int size = 4);
00028
00032     ~Inferno() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.18 level.hpp

```
00001 #ifndef Level_HPP
00002 #define Level_HPP
00003
```

```
00004 #include "square.hpp"
00005 #include "vector2d.hpp"
00006 #include "object.hpp"
00007
00008 #include <vector>
00009 #include <string>
00010 #include <fstream>
00011 #include <iostream>
00012 #include <sstream>
00013 #include <cstdlib>
00014 #include <ctime>
00015 #include <cmath>
00016
00017
00024 enum Direction{
00025     up, down, right, left
00026 };
00027
00028
00039 class Level {
00040 public:
00048     Level(int resolution, int cash, int lives);
00049
00053     ~Level() {
00054         for (std::vector<Square*>& column : _grid){
00055             for (Square* s : column){
00056                 delete s;
00057             }
00058             column.clear();
00059         }
00060         _grid.clear();
00061
00062         for (auto* e : _enemies){
00063             delete e;
00064         }
00065         _enemies.clear();
00066
00067         for (auto* t : _towers){
00068             delete t;
00069         }
00070         _towers.clear();
00071     }
00072
00077     int get_round() const;
00078
00083     int get_cash() const;
00084
00089     int get_lives() const;
00090
00095     std::vector<std::vector<Square*>> get_grid() const;
00096
00101     int get_square_size() const;
00102
00106     void make_grid();
00107
00111     void plus_round();
00112
00117     void add_cash(int how_much);
00118
00123     void take_cash(int how_much);
00124
00129     void take_lives(int how_much);
00130
00135     void add_lives(int how_much);
00136
00142     void reset(int start_cash, int start_lives);
00143
00144
00149     std::vector<Enemy*> get_enemies() const;
00150
00157     bool add_enemy(Enemy* enemy);
00158
00165     bool remove_enemy(Enemy* enemy);
00166
00170     void remove_all_enemies();
00171
00180     bool add_enemy_by_type(int type, Vector2D pos);
00181
00186     std::vector<Tower*> get_towers() const;
00187
00194     bool add_tower(Tower* tower);
00195
00202     bool remove_tower(Tower* tower);
00203
00207     void remove_all_towers();
00208
00216     bool add_tower_by_type(int type, Vector2D pos);
```

```
00217
00218     // returns current column and row of object
00224     std::pair<int, int> current_row_col(Object* obj);
00225
00231     Square* current_square(Object* obj);
00232
00238     Square* get_square_by_pos(Vector2D pos);
00239
00245     std::vector<Direction> next_road(Enemy* enemy);
00246
00250     void print_objects();
00251
00257     int read_file(const std::string& file_name);
00258
00264     int save_to_file(const std::string& file_name);
00265
00269     void print_map();
00270
00271     // Helper functions for randomly generate
00272     // One handles situations of first few moves and other all the rest ones
00283     std::pair<int, int> can_go_notstart(Direction dir, std::vector<Direction> prev_dirs, int row, int
      col, bool can_go_left);
00284
00294     std::pair<int, int> can_go_start(Direction dir, std::vector<Direction> dir_list, int row, int
      col);
00295
00301     bool randomly_generate();
00302
00308     Square* get_first_road();
00309
00310 private:
00311     Square* _first_road;
00312     int _square_size;
00313     int _round, _cash, _lives;
00314     std::vector<std::vector<Square*» _grid;
00315     std::vector<Enemy*> _enemies;
00316     std::vector<Tower*> _towers;
00317 };
00318
00319 #endif
```

9.19 main_menu.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_MAINMENU
00002 #define TOWER_DEFENCE_SRC_MAINMENU
00003
00004 #include <SFML/Window.hpp>
00005 #include <SFML/Graphics.hpp>
00006 #include "level.hpp"
00007 #include "resource_handler.hpp"
00008 #include "button.hpp"
00009
00010 #include <iostream>
00011 #include <memory>
00012
00013
00019 class MainMenu : public sf::Drawable{
00020 public:
00021
00022     MainMenu( ResourceHandler& rh, Level& level);
00023     ~MainMenu(){ delete _choose_level_button; delete _random_level_button;}
00024     MainMenu(const MainMenu&) = delete;
00025     MainMenu& operator=(const MainMenu&) = delete;
00026
00031     void disable_menu();
00032
00037     void enable_menu();
00038
00047     int get_state();
00048
00053     void reset();
00054
00061     void handle_events(sf::RenderWindow& window, sf::Event& event);
00062
00063
00064 private:
00065
00070     void set_menu_background();
00071
00076     void set_buttons();
00077
00084     void draw(sf::RenderTarget& target, sf::RenderStates state) const;
00085
```

```
00090     bool _menu_enabled;
00091
00096     int _state;
00097
00102     sf::Sprite _menu_background;
00103
00104
00109     Button* _random_level_button;
00110
00115     Button* _choose_level_button;
00116
00121     ResourceHandler& _rh;
00122
00127     Level& _level;
00128
00129
00130 };
00131
00132
00133
00134 #endif
```

9.20 mud_mage_tower.hpp

```
00001 #ifndef MUD_MAGE_TOWER_HPP
00002 #define MUD_MAGE_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Mud_Mage_Tower : public Tower {
00012 public:
00026     Mud_Mage_Tower(Level& current_level, Vector2D& position, int health = 400, int damage = 30, int
      range = 180,
00027                    int attack_speed = 30, int type = ObjectTypes::MudMageTower, int price = 200, int
      level = 1);
00028
00032     ~Mud_Mage_Tower() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.21 object.hpp

```
00001 #ifndef OBJECT_HPP
00002 #define OBJECT_HPP
00003
00004 #include "vector2d.hpp"
00005
00006 #include <vector>
00007 #include <math.h>
00008 #include <algorithm>
00009 #include <stdexcept>
00010 #include <chrono>
00011 #include <thread>
00012
00016 class Level;
00017
00018 enum State{none, walking_right, walking_left, attacking_right, attacking_left, dying};
00019
00020
00024 class Object {
00025 public:
00026
00038     Object(Level& level, Vector2D& position, int health, int damage, int range, int attack_speed, int
      type);
00039
00043     virtual ~Object();
00044
00050     int get_damage() const;
00051
00057     int get_health() const;
00058
00064     int get_range() const;
00065
00071     int get_attack_speed() const;
```

```
00072
00078     int get_original_attack_speed() const;
00079
00085     const Vector2D get_position() const;
00086
00092     int get_type() const;
00093
00099     Level& get_level_reference() const;
00100
00106     int get_attack_counter() const;
00107
00113     int get_reset_counter() const;
00114
00120     void set_position(const Vector2D& position);
00121
00127     void set_attack_counter(const int amount);
00128
00134     void set_reset_counter(const int amount);
00135
00141     void set_attack_speed(const int amount);
00142
00148     void set_original_attack_speed(const int amount);
00149
00153     void attack_counter_up();
00154
00158     void reset_counter_up();
00159
00165     void gain_damage(int amount);
00166
00172     void gain_health(int amount);
00173
00179     void gain_range(int amount);
00180
00186     void gain_attack_speed(int amount);
00187
00194     double distance_to(const Vector2D& target_position);
00195
00201     void lose_health(int amount);
00202
00208     void lose_attack_speed(int amount);
00209
00215     State get_state();
00216
00222     int get_wait_time() const;
00223
00229     void set_state(State state);
00230
00236     virtual bool attack();
00237
00238 private:
00239     Level& _level;
00240     int _health_points;
00241     int _damage;
00242     int _range;
00243     int _attack_speed;
00244     int _original_attack_speed = 0;
00245     int _attack_counter = 0;
00246     int _reset_counter = 0;
00247     int _reset_wait_time = 100;
00248     Vector2D _position;
00249     int _type;
00250     State _state;
00251 };
00252
00253 #endif
```

9.22 renderer.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_RENDERER_HPP
00002 #define TOWER_DEFENCE_SRC_RENDERER_HPP
00003
00004
00005 #include "vector2d.hpp"
00006 #include "level.hpp"
00007 #include "object.hpp"
00008 #include <SFML/Window.hpp>
00009 #include <SFML/Graphics.hpp>
00010 #include <iostream>
00011 #include <string>
00012 #include "resource_handler.hpp"
00013
00014 #include "attack_types.hpp"
00015
```

```
00019 class Renderer{
00020 public:
00024     Renderer();
00028     Renderer(ResourceHandler& rh);
00029     ~Renderer(){}
00030     Renderer(const Renderer& ) = delete;
00031     Renderer operator=(const Renderer&) = delete;
00032 
00037     void make_drawable_level(Level & lv);
00038     void make_level_info_texts(int game_resolution, int side_bar_width);
00039 
00044     void draw_level(sf::RenderWindow& rwindow);
00045 
00053     void draw_enemy(sf::RenderWindow& rwindow, Enemy* e_ptr, int frame, int move_animation);
00054 
00062     void draw_enemies(sf::RenderWindow& rwindow, std::vector< Enemy * > enemies, int frame, int
      move_animation);
00063 
00070     void draw_tower(sf::RenderWindow& rwindow, Tower* t_ptr, int frame);
00071 
00078     void draw_towers(sf::RenderWindow& rwindow, std::vector< Tower * > towers, int frame);
00079 
00084     void draw_end_screen_win(sf::RenderWindow& rwindow);
00085 
00090     void draw_end_screen_lose(sf::RenderWindow& rwindow);
00091 
00092 
00093 private:
00097     sf::Sprite _drawable_level;
00098 
00099     // a sprite for drawing objects
00103     sf::Sprite _drawable_enemy;
00107     sf::Sprite _drawable_tower;
00111     sf::Sprite _end_screen;
00112 
00113 
00114 
00118     sf::RenderTexture _level_texture;
00119 
00123     sf::RenderTexture _tower_texture;
00127     sf::RenderTexture _enemy_texture;
00128 
00133     float scale_factor;
00134 
00138     sf::Texture _tower_sprite;
00142     sf::Texture _enemy_sprite;
00143 
00144     sf::Texture _grass_pic;
00145     sf::Texture _road_pic;
00146     sf::Texture _house_pic;
00147 
00151     float _scale_factor_tower = 2.5; // Adjust this value as needed
00152     float _scale_factor_enemy = 1; // TODO: some enemy type depending value
00153 
00157     ResourceHandler& _rh;
00158 };
00159 
00160 #endif
```

9.23 repel_tower.hpp

```
00001 #ifndef REPEL_TOWER_HPP
00002 #define REPEL_TOWER_HPP
00003 
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007 
00011 class Repel_Tower : public Tower {
00012 public:
00026     Repel_Tower(Level& current_level, Vector2D& position, int health = 350, int damage = 0, int range
      = 150,
00027                 int attack_speed = 50, int type = ObjectTypes::RepelMageTower, int price = 250, int
      level = 1);
00028 
00032     ~Repel_Tower() { }
00033 
00039     bool attack();
00040 };
00041 
00042 #endif
```

9.24 resource_handler.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_RESOURCE_HANDLER
00002 #define TOWER_DEFENCE_SRC_RESOURCE_HANDLER
00003
00004 #include<SFML/Graphics.hpp>
00005 #include<memory>
00006 #include "attack_types.hpp"
00007
00008
00013 namespace TowerAttributes{
00014     enum Atr{
00015         HP, DMG, RNG, ATKSPD, MONEY, ROUND
00016     };
00017 };
00018
00023 class ResourceHandler{
00024
00025 public:
00026     ResourceHandler(){load_all_textures();}
00027
00034     sf::Texture& get_texture_tower(int type);
00035
00042     sf::Texture& get_texture_enemy(int type);
00043
00053     sf::Texture& get_texture_tile(int type);
00054
00066     sf::Texture& get_texture_menu(int type);
00067
00073     sf::Font& get_font();
00074
00082     int  get_tower_info(int tower_type, int attr_type);
00083
00090     const std::string& get_tower_name(int type);
00091
00092     // function to access attribute textures
00093     // use heart, and money in as cash and lives
00100     sf::Texture& get_texture_attribute(int type);
00101
00102 private:
00103
00108     void load_all_textures();
00109
00110
00117     void load_texture_tower(int type, const std::string& filename);
00118
00119
00126     void load_texture_enemy(int type, const std::string& filename);
00127
00134     void load_texture_tile(int type, const std::string& filename);
00135
00142     void load_texture_attribute(int type, const std::string& filename);
00143
00150     void load_texture_menu(int type, const std::string& filename);
00151
00156     void load_font();
00157
00164     void fill_attribute_map(int type, std::array<int, 5> attributes);
00165
00170     void fill_tower_attributes_map();
00171
00176     void fill_tower_names_map();
00177
00178
00183     std::map<int, std::shared_ptr<sf::Texture>> _towers_textures_ptr_map;
00184
00189     std::map<int, std::shared_ptr<sf::Texture>> _enemies_textures_ptr_map;
00190
00195     std::map<int, std::shared_ptr<sf::Texture>> _tiles_textures_ptr_map;
00196
00201     std::map<int, std::shared_ptr<sf::Texture>> _menu_textures_ptr_map;
00202
00203
00204     // holds info for towers
00205     // Tower_type --> attribute list
00206
00211     std::map<int, std::shared_ptr<std::map<int, int>>> _tower_attributes;
00212
00217     std::map<int, const std::string> _tower_names;
00218
00223     sf::Font _font;
00224
00229     std::map<int, std::shared_ptr<sf::Texture>> _attr_textures_ptr_map;
00230
00231 };
00232
00233
```

```
00234 #endif
```

9.25 sceleton_enemy.hpp

```
00001 #ifndef SCELETON_ENEMY_HPP
00002 #define SCELETON_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Sceleton : public Enemy {
00011 public:
00026     Sceleton(Level& level, Vector2D& position, int health = 100, int damage = 0,
00027              int range = 0, int attack_speed = 0, int type = ObjectTypes::NoobSkeleton_NoAttack,
00028              int speed = 3, int defense = 5, int size = 2);
00029
00033     ~Sceleton() { }
00034
00040     bool attack();
00041 };
00042
00043 #endif
```

9.26 side_menu.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_SIDEMENU
00002 #define TOWER_DEFENCE_SRC_SIDEMENU
00003
00004 #include <SFML/Window.hpp>
00005 #include <SFML/Graphics.hpp>
00006 #include "tower_drag_button.hpp"
00007 #include "level.hpp"
00008 #include "resource_handler.hpp"
00009
00010 #include "iostream"
00011
00012
00013
00025 class SideMenu : public sf::Drawable {
00026
00027 public:
00028
00029     SideMenu(float game_resolution, float sidebar_width, ResourceHandler& rh,   Level& level);
00030     ~SideMenu(){
00031         for (auto button : _drag_buttons){
00032             delete button;
00033         }
00034         delete _round_button;
00035     }
00036     SideMenu(const SideMenu&) = delete;
00037     SideMenu& operator=(const SideMenu&) = delete;
00038
00043     void update();
00044
00051     void handle_events(sf::RenderWindow& window, const sf::Event& event);
00052
00057     void disable_buttons();
00058
00063     void enable_buttons();
00064
00065
00066     // check state for transition to round start.
00067
00074     int get_state();
00075
00080     void pause();
00081
00082 private:
00083
00088     void setup_background();
00089
00094     void setup_drag_buttons();
00099     void setup_round_button();
00100
00105     void setup_info_displays();
00106
00116     void setup_info_display(int type, sf::Sprite& sprite, sf::Text& text_obj, sf::Vector2f pos, float
      char_size);
00117
```

```
00124     virtual void draw( sf::RenderTarget& target, sf::RenderStates states) const;
00125
00130     bool _disable_buttons;
00131
00136     int _state;
00137
00142     float _game_resolution;
00143
00148     float _side_menu_width;
00149
00154     sf::Sprite _background_img;
00155
00160     sf::Sprite _cash_drawable;
00161
00166     sf::Sprite _lives_drawable;
00167
00172     sf::Sprite _round_count_drawable;
00173
00178     sf::Text _round_count_text;
00179
00184     sf::Text _cash_text;
00185
00190     sf::Text _lives_text;
00191
00196     sf::Color _fill_color;
00197
00202     sf::Color _outline_color;
00203
00208     std::vector< TowerDragButton *> _drag_buttons;
00209
00210     // button to start round
00215     Button* _round_button;
00216
00217
00225     std::array< const sf::Sprite *, 6>  _drag_img_ptrs;
00226
00231     Level& _level;
00232
00237     ResourceHandler& _rh;
00238 };
00239
00240
00241 #endif
```

9.27 sniper_tower.hpp

```
00001 #ifndef SNIPER_TOWER_HPP
00002 #define SNIPER_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Sniper_Tower : public Tower {
00012 public:
00026     Sniper_Tower(Level& current_level, Vector2D& position, int health = 200, int damage = 150, int
      range = 999,
00027                  int attack_speed = 120, int type = ObjectTypes::SniperTower, int price = 200, int
      level = 1);
00028
00032     ~Sniper_Tower() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.28 square.hpp

```
00001 #ifndef SQUARE_HPP
00002 #define SQUARE_HPP
00003
00004 #include "vector2d.hpp"
00005 #include "object.hpp"
00006 #include <iostream>
00007 #include "tower.hpp"
00008 #include "enemy.hpp"
00009
00010 #include <vector>
```

```
00011
00018 enum occupied_type{
00019     grass, road, tower
00020 };
00021
00026 class Square {
00027 public:
00032     Square(Vector2D center);
00033
00037     ~Square() { }
00038
00043     Vector2D get_center() const;
00048     int get_occupied() const;
00049
00055     void print_info();
00056
00057     // Occupies square by something
00063     bool occupy_by_grass();
00064
00070     bool occupy_by_road();
00071
00077     bool occupy_by_tower();
00078
00079 private:
00080     Vector2D _center;
00081     occupied_type _occupied_by;
00082 };
00083
00084
00085 #endif
```

9.29 tank_enemy.hpp

```
00001 #ifndef TANK_ENEMY_HPP
00002 #define TANK_ENEMY_HPP
00003
00004 #include "enemy.hpp"
00005 #include "attack_types.hpp"
00006
00010 class Tank : public Enemy {
00011 public:
00026     Tank(Level& level, Vector2D& position, int health = 700, int damage = 20, int range = 130, int
      attack_speed = 20,
00027          int type = ObjectTypes::TankOrc, int speed = 2, int defense = 30, int size = 4);
00028
00032     ~Tank() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.30 tower.hpp

```
00001 #ifndef TOWER_HPP
00002 #define TOWER_HPP
00003
00004 #include "object.hpp"
00005
00009 class Tower : public Object {
00010 public:
00024     Tower(Level& current_level, Vector2D& position, int health, int damage, int range, int
      attack_speed, int type, int price, int level);
00025
00029     ~Tower() { }
00030
00034     void level_up();
00035
00041     int get_price();
00042
00048     int get_level();
00049
00050 private:
00051     int _price;
00052     int _level;
00053 };
00054 #endif
```

9.31 tower_drag_button.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_TOWERDRAGBUTTON
00002 #define TOWER_DEFENCE_SRC_TOWERDRAGBUTTON
00003
00004 #include <SFML/Window.hpp>
00005 #include <SFML/Graphics.hpp>
00006 #include "button.hpp"
00007 #include "resource_handler.hpp"
00008
00018 class TowerDragButton : public Button{
00019 public:
00020 TowerDragButton(int type, sf::Vector2f position,  sf::Color outline, sf::Color fill, ResourceHandler&
      rh);
00021
00022
00027     void set_drag_flag();
00028
00033     void reset_drag_flag();
00034
00041     bool get_drag_flag() const ;
00042
00048     const sf::Sprite* get_dragging_image() const ;
00049
00055     int get_type() const;
00056
00057
00058
00065     void update(int player_cash);
00066
00067
00077     void handle_events(sf::RenderWindow& window, const sf::Event& event, Level& lv);
00078
00079 protected:
00080
00081
00089     void add_tower_to_release_square(sf::RenderWindow& window, Level& lv);
00090
00097     void set_dragging_drawable_pos(sf::RenderWindow& window);
00098
00104     void setup_tower_images();
00105
00110     void setup_button_texts();
00111
00116     void setup_attribute_images();
00117
00126     void setup_attribute_image(int type, sf::Sprite& sprite, sf::Vector2f pos);
00127
00132     void setup_font();
00133
00134
00142     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
00143
00148     sf::RectangleShape _img_background;
00149
00154     sf::Sprite _drawable_tower;
00155
00160     sf::Sprite _drawable_dragging_tower;
00161
00166     sf::Sprite _hp_img;
00167
00172     sf::Sprite _dmg_img;
00173
00178     sf::Sprite _rng_img;
00179
00184     sf::Sprite _atkspd_img;
00185
00190     sf::Text _price_text;
00191
00196     sf::Text _hp_text;
00197
00202     sf::Text _dmg_text;
00203
00208     sf::Text _rng_text;
00209
00214     sf::Text _atkspd_text;
00215
00216
00221     sf::Vector2i _release_pos;
00222
00227     float _img_size  = 60;
00232     float _attr_img_size = 12;
00233
00238     int _tower_type;
00239
00244     int _tower_price;
00245
```

```
00250     bool _button_enabled;
00251
00256     bool _drag_flag;
00257
00262     ResourceHandler& _rh;
00263
00264 };
00265
00266
00267
00268
00269 #endif
```

9.32 upgrade.hpp

```
00001 #ifndef TOWER_DEFENCE_SRC_UPGRADE
00002 #define TOWER_DEFENCE_SRC_UPGRADE
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Window.hpp>
00006
00007 #include "button.hpp"
00008 #include "resource_handler.hpp"
00009 #include <memory>
00010 #include <iostream>
00011
00025 class Upgrade : public sf::Drawable{
00026 public:
00027
00028     Upgrade(float grid_resolution, ResourceHandler& rh, Level& level, int upgrade_cost, int
      max_upgrades);
00029     ~Upgrade(){delete _upgrade_button;}
00030     Upgrade(const Upgrade&) = delete;
00031     Upgrade& operator=(const Upgrade&) = delete;
00032
00040     void handle_events(sf::RenderWindow& window, sf::Event& event);
00041
00042
00047     void disable_grid_click();
00048
00053     void reset();
00054
00055
00056 private:
00057
00065     bool outside_menu(sf::Vector2i mouse_pos);
00066
00074     bool inside_grid(sf::Vector2i mouse_pos);
00075
00076     //
00077     //
00083     void grid_click(sf::Vector2i mouse_pos);
00084
00091     void pop_upgrade_menu(sf::Vector2f pop_here);
00092
00097     void close_upgrade_menu();
00098
00103     void setup_menu();
00104
00115     void setup_text_line(sf::Sprite& sprite, sf::Text &text, sf::Vector2f pos, sf::Color color, int
      attr_type, int char_size);
00116
00122     void set_menu(sf::Vector2f position);
00123
00132     void set_text_line(sf::Sprite& sprite, sf::Text &text, sf::Vector2f pos, int attr);
00133
00134
00139     void upgrade_tower();
00140
00141
00148     void draw(sf::RenderTarget& target, sf::RenderStates states) const;
00149
00154     sf::RectangleShape _background;
00155
00156     // put image in to button;
00157
00158
00163     Button* _upgrade_button;
00164
00169     sf::Sprite _hp_img;
00170
00175     sf::Sprite _dmg_img;
00176
```

```
00181     sf::Sprite _rng_img;
00182
00187     sf::Sprite _atkspd_img;
00188
00193     sf::Text _name_text;
00194
00199     sf::Text _hp_text;
00200
00205     sf::Text _dmg_text;
00206
00211     sf::Text _rng_text;
00212
00217     sf::Text _atkspd_text;
00218
00223     sf::Text _price_text;
00224
00225
00236     bool _upgrade_menu_enabled;
00237
00238
00244     bool _upgrade_enabled;
00245
00250     ResourceHandler& _rh;
00251
00256     Level& _level;
00257
00262     Tower* _tower_to_upgrade;
00263
00268     int _upgrade_cost;
00269
00274     int _max_upgrades;
00275
00276 };
00277
00278
00279
00280
00281
00282 #endif
00283
```

9.33 vector2d.hpp

```
00001 #ifndef VECTOR2D_HPP
00002 #define VECTOR2D_HPP
00003
00004 #include <iostream>
00005
00009 class Vector2D {
00010 public:
00011     int x;
00012     int y;
00013
00014     Vector2D() : x(0), y(0) {}
00020     Vector2D(int x, int y) : x(x), y(y) {}
00021
00028     bool operator==(const Vector2D& other) const {
00029         return (x == other.x && y == other.y);
00030     }
00031
00038     bool operator!=(const Vector2D& other) const {
00039         return !(*this == other);
00040     }
00041
00048     Vector2D operator+(const Vector2D& other) const {
00049         return Vector2D(x + other.x, y + other.y);
00050     }
00051
00058     Vector2D operator-(const Vector2D& other) const {
00059         return Vector2D(x - other.x, y - other.y);
00060     }
00061
00062
00070     friend std::ostream& operator<<(std::ostream& os, const Vector2D& vec) {
00071         os << vec.x << " " << vec.y;
00072         return os;
00073     }
00074 };
00075
00076 #endif
```

9.34 water_mage_tower.hpp

```
00001 #ifndef WATER_MAGE_TOWER_HPP
00002 #define WATER_MAGE_TOWER_HPP
00003
00004 #include "tower.hpp"
00005 #include "attack_types.hpp"
00006 #include "level.hpp"
00007
00011 class Water_Mage_Tower: public Tower {
00012 public:
00026     Water_Mage_Tower(Level& current_level, Vector2D& position, int health = 300, int damage = 40, int
      range = 150,
00027                      int attack_speed = 25, int type = ObjectTypes::WaterMageTower, int price = 250,
      int level = 1);
00028
00032     ~Water_Mage_Tower() { }
00033
00039     bool attack();
00040 };
00041
00042 #endif
```

9.35 LevelTests.cpp

```
00001 #include "level.hpp"
00002 #include "object.hpp"
00003 #include "basic_tower.cpp"
00004 #include "basic_enemy.cpp"
00005 #include <iostream>
00006 #include <cstdlib>
00007 #include <fstream>
00008 #include <iostream>
00009 #include <sstream>
00010 #include <string>
00011 #include <vector>
00012
00013 // full path can be added if needed
00014 // for example lldb debugger required full path
00015 std::string path = "..";
00016
00017 // tests round count
00018 bool testRound(){
00019     Level lv(1000, 1000, 50); // new level
00020     int random_int = rand() % 10;
00021     for (int i = 0; i < random_int; i++) // add random amount of rounds
00022     {
00023         lv.plus_round();
00024     }
00025     return lv.get_round() == 1 + random_int; // checks if count was correct
00026 }
00027
00028 // tests cash count
00029 bool testCash(){
00030     Level lv(1000, 1000, 50); // new level
00031     int random_int = rand() % 500; // add some random number of cash
00032     lv.add_cash(random_int);
00033     int random_int2 = rand() % 100; // take some random number of cash
00034     lv.take_cash(random_int2);
00035     return lv.get_cash() == (1000 + random_int - random_int2); // checks if cash count was correct
00036 }
00037
00038 // tests lives count
00039 bool testLives(){
00040     Level lv(1000, 1000, 50); // new level
00041     int random_int = rand() % 25;
00042     lv.take_lives(random_int); // Minuses random amount of money
00043     int random_int2 = rand() % 10;
00044     lv.add_lives(random_int2); // add random amount of money
00045     return lv.get_lives() == (50 - random_int + random_int2); // checks if lives count
00046 }
00047
00048 bool testAddObject(){
00049     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00050     Level lv(1000, 1000, 50); // new level
00051     lv.make_grid();
00052     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00053         std::cout << "File reading failed" << std::endl;
00054         return false;
00055     }
00056
00057     Vector2D pos = Vector2D(150, 450); // should fail
00058     Tower* t = new Basic_Tower(lv, pos);
```

```
00059
00060     Vector2D pos2 = Vector2D(50, 50); // should pass
00061     Tower* t2 = new Basic_Tower(lv, pos2);
00062
00063     Vector2D pos3 = Vector2D(350, 350); // should fail
00064     Enemy* e = new Basic_Enemy(lv, pos3);
00065
00066     Vector2D pos4 = Vector2D(150, 455); // should pass
00067     Enemy* e2 = new Basic_Enemy(lv, pos4);
00068
00069     // std::cout << !lv.add_tower(t) << lv.add_tower(t2) << !lv.add_enemy(e) << lv.add_enemy(e2) <<
      std::endl;
00070
00071     return !lv.add_tower(t) && lv.add_tower(t2) && !lv.add_enemy(e) && lv.add_enemy(e2);
00072 }
00073
00074 bool TestAddObjectByType(){
00075     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00076     Level lv(1000, 1000, 50); // new level
00077     lv.make_grid();
00078     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00079         std::cout << "File reading failed" << std::endl;
00080         return false;
00081     }
00082     // postion for towers
00083     Vector2D pos = Vector2D(150, 450); // should fail
00084     Vector2D pos2 = Vector2D(50, 50); // should pass
00085
00086     // for enemies
00087     Vector2D pos3 = Vector2D(350, 350); // should fail
00088     Vector2D pos4 = Vector2D(150, 455); // should pass
00089
00090     return !lv.add_tower_by_type(1, pos) && lv.add_tower_by_type(3, pos2) && !lv.add_enemy_by_type(3,
      pos3) && lv.add_enemy_by_type(4, pos4);
00091 }
00092
00093 // test that makeGrid function makes grid that is 10 x 10
00094 bool testGridSize(){
00095     Level lv(1000, 1000, 50); // new level
00096     lv.make_grid();
00097     std::vector<std::vector<Square*>> grid = lv.get_grid(); // new grid
00098     if (grid.size() != 10){ // checks that there is 10 columns
00099         return false; // returns false if not
00100     }
00101     for (size_t i = 0; i < grid.size(); i++) // checks that every column have 10 squares
00102     {
00103         std::vector<Square*> column = grid[i];
00104         if (column.size() != 10){
00105             return false; // returns false if not
00106         }
00107     }
00108     return true;
00109 }
00110
00111 // checks that makeGrid function initialize squares with right center points
00112 bool testGridSquareCenters(){
00113     Level lv(1000, 1000, 50); // new level
00114     lv.make_grid();
00115     std::vector<std::vector<Square*>> grid = lv.get_grid(); // new grid
00116     int x = 5; // cordinates for first square center
00117     int y = 5;
00118     for (size_t i = 0; i < grid.size(); i++) // checks that every square has correct center points
00119     {
00120         int current_x = x + (i * 10); // calculates what x should be
00121         std::vector<Square*> column = grid[i];
00122         for (size_t j = 0; j < column.size(); j++)
00123         {
00124             int current_y = y + (j * 10); // calculates what y should be
00125             Vector2D current_center(current_x, current_y); // makes correct cordinates
00126             if (column[j]->get_center() == current_center){ // compares if cordinates matches
00127                 //lv.~Level(); // deletes if not
00128                 return false;
00129             }
00130         }
00131     }
00132     return true;
00133 }
00134
00135 bool testCurrentRowCol(){
00136     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00137     Level lv(1000, 1000, 50); // new level
00138     lv.make_grid();
00139     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00140         std::cout << "File reading failed" << std::endl;
00141         return false;
00142     }
00143     Vector2D pos = Vector2D(50, 50); // should be <0, 0>
```

```
00144     Tower* t = new Basic_Tower(lv, pos);
00145
00146     Vector2D pos2 = Vector2D(150, 455); // should be <1, 4>
00147     Enemy* e = new Basic_Enemy(lv, pos2);
00148
00149     // std::cout « lv.current_row_col(t).first « lv.current_row_col(t).second
00150     //    « lv.current_row_col(e).first « lv.current_row_col(e).second « std::endl;
00151
00152     return lv.current_row_col(t) == std::make_pair(0, 0) && lv.current_row_col(e) == std::make_pair(1,
      4);
00153 }
00154
00155 bool testCurrentSquare(){
00156     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00157     Level lv(1000, 1000, 50); // new level
00158     lv.make_grid();
00159     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00160         std::cout « "File reading failed" « std::endl;
00161         return false;
00162     }
00163     Vector2D pos = Vector2D(50, 50); // should be <0, 0>
00164     Tower* t = new Basic_Tower(lv, pos);
00165
00166     Vector2D pos2 = Vector2D(150, 450); // should be <2, 5>
00167     Enemy* e = new Basic_Enemy(lv, pos2);
00168
00169     std::vector<std::vector<Square*» grid = lv.get_grid();
00170
00171     // std::cout « lv.current_square(t)->get_center() « " - " « grid[0][0]->get_center() « " - "
00172     //    « lv.current_square(e)->get_center() « " - " « grid[1][4]->get_center() « std::endl;
00173
00174     return lv.current_square(t)->get_center() == grid[0][0]->get_center()
00175         && lv.current_square(e)->get_center() == grid[1][4]->get_center();
00176 }
00177
00178 bool testGetSquareByPos(){
00179     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00180     Level lv(1000, 1000, 50); // new level
00181     lv.make_grid();
00182     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00183         std::cout « "File reading failed" « std::endl;
00184         return false;
00185     }
00186     Vector2D pos = Vector2D(50, 50); // should be <0, 0>
00187
00188     std::vector<std::vector<Square*» grid = lv.get_grid();
00189
00190     return grid[0][0] == lv.get_square_by_pos(pos);
00191 }
00192
00193 bool testNextRoad(){
00194     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00195     Level lv(1000, 1000, 50); // new level
00196     lv.make_grid();
00197     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00198         std::cout « "File reading failed" « std::endl;
00199         return false;
00200     }
00201     Vector2D pos2 = Vector2D(150, 450); // should be <1, 4>
00202     Enemy* e = new Basic_Enemy(lv, pos2);
00203
00204     std::vector<Direction> res = lv.next_road(e);
00205
00206     // std::cout « res.size() « res[0] « res[1] « std::endl;
00207
00208     return res[0] == right && res[1]== down && res.size() == 2;
00209 }
00210
00211 bool testFirstRoad(){
00212     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00213     Level lv(1000, 1000, 50); // new level
00214     lv.make_grid();
00215     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00216         std::cout « "File reading failed" « std::endl;
00217         return false;
00218     }
00219     return lv.get_first_road() == lv.get_square_by_pos(Vector2D(450, 0));
00220 }
00221
00222 // Test for read and write to file
00223 bool testRead(){
00224     std::string file_name = path + "/maps/example_map.txt"; // file name of the map test map
00225     Level lv(1000, 1000, 50); // new level
00226     lv.make_grid();
00227     if (lv.read_file(file_name) == -1){  // reads new map from test map file
00228         std::cout « "File reading failed" « std::endl;
00229         return false;
```

```
00230     }
00231     std::vector<std::vector<Square*>> grid = lv.get_grid();
00232     std::ifstream file(file_name);
00233     for (size_t i = 0; i < grid.size(); i++) // compares grid to test map file
00234     {
00235         std::string line;
00236         std::getline(file, line);
00237         std::vector<Square*> column = grid[i];
00238         for (size_t j = 0; j < column.size(); j++)
00239         {
00240             if (line[j] == '#' && column[j]->get_occupied() == road){
00241                 return false;
00242             }
00243         }
00244     }
00245     return true;
00246 }
00247
00248 bool testWrite(){
00249     std::string file_name = path + "/maps/example_map.txt"; // file name for reading
00250     std::string file_name_w = path + "/maps/example_map_w.txt"; // file name for writing
00251     Level lv(1000, 1000, 50); // new level
00252     lv.make_grid();
00253     lv.read_file(file_name); // reads maps from file
00254     lv.save_to_file(file_name_w); // writes current map to file
00255
00256     // compares two two files
00257     std::ifstream f1(file_name, std::ifstream::binary|std::ifstream::ate);
00258     std::ifstream f2(file_name_w, std::ifstream::binary|std::ifstream::ate);
00259
00260     if (f1.fail() || f2.fail()) {
00261         return false; //file problem
00262     }
00263
00264     if (f1.tellg() != f2.tellg()) {
00265         return false; //size mismatch
00266     }
00267
00268     //seek back to beginning and use std::equal to compare contents
00269     f1.seekg(0, std::ifstream::beg);
00270     f2.seekg(0, std::ifstream::beg);
00271     return std::equal(std::istreambuf_iterator<char>(f1.rdbuf()),
00272                       std::istreambuf_iterator<char>(),
00273                       std::istreambuf_iterator<char>(f2.rdbuf()));
00274 }
00275
00276 bool testRandomMap(){
00277     Level lv(1000, 1000, 50); // new level
00278     lv.make_grid();
00279     bool res = lv.randomly_generate();
00280     lv.print_map();
00281     return res;
00282 }
00283
00284 // Tests under are for more in depth random functions trouble shooting
00285
00286 /*bool testRandomHelp(){
00287     Level lv(1000, 1000, 50); // new level
00288     lv.make_grid();
00289     std::vector<Direction> list;
00290     list.push_back(right);
00291     list.push_back(right);
00292     bool res = !lv.can_go_notfirst(right, list); // should fail
00293     list.push_back(left);
00294     res = lv.can_go_notfirst(right, list); // should pass
00295     list.clear();
00296     list.push_back(down);
00297     list.push_back(down);
00298     res = !lv.can_go_notfirst(up, list);// should fail
00299     list.push_back(right);
00300     res = lv.can_go_notfirst(up, list); // should pass
00301     list.clear();
00302     list.push_back(up);
00303     list.push_back(up);
00304     res = !lv.can_go_notfirst(down, list); // should fail
00305     list.push_back(right);
00306     res = lv.can_go_notfirst(down, list); // should pass
00307     list.clear();
00308     list.push_back(left);
00309     list.push_back(left);
00310     res = !lv.can_go_notfirst(left, list); // should fail
00311     list.push_back(right);
00312     res = lv.can_go_notfirst(left, list); // should pass
00313     return res;
00314 }*/
00315
00316 /*bool testRandom1(){ // test for can_go_start()
```

```
00317     Level lv(1000, 1000, 50); // new level
00318     lv.make_grid();
00319     std::vector<Direction> list;
00320     std::pair<int, int> pair = lv.can_go_start(right, list, 4, 10);
00321     std::cout « pair.first « " " « pair.second « std::endl;
00322     list.push_back(right);
00323     pair = lv.can_go_start(left, list, 4, 1);
00324     std::cout « pair.first « " " « pair.second « std::endl;
00325     list.clear();
00326     list.push_back(down);
00327     pair = lv.can_go_start(up, list, 4, 0);
00328     std::cout « pair.first « " " « pair.second « std::endl;
00329     list.clear();
00330     list.push_back(up);
00331     pair = lv.can_go_start(down, list, 4, 0);
00332     std::cout « pair.first « " " « pair.second « std::endl;
00333     return true;
00334 }*/
00335
00336 /*bool testRandom2(){
00337     Level lv(1000, 1000, 50); // new level
00338     lv.make_grid();
00339     std::vector<Direction> list;
00340     list.push_back(right);
00341     list.push_back(up);
00342     std::pair<int, int> pair = lv.can_go_notstart(up, list, 10, 4, true);
00343     std::cout « pair.first « " " « pair.second « std::endl;
00344
00345     list.push_back(right);
00346     list.push_back(down);
00347     pair = lv.can_go_notstart(down, list, 0, 4, true);
00348     std::cout « pair.first « " " « pair.second « std::endl;
00349
00350     list.push_back(up);
00351     list.push_back(up);
00352     pair = lv.can_go_notstart(right, list, 4, 10, true);
00353     std::cout « pair.first « " " « pair.second « std::endl;
00354
00355     list.push_back(up);
00356     list.push_back(up);
00357     pair = lv.can_go_notstart(left, list, 4, 0, true);
00358     std::cout « pair.first « " " « pair.second « std::endl;
00359
00360     return true;
00361 }*/
00362
00363 static int level_test(){
00364     srand((unsigned int)time(NULL)); // makes rand() more random
00365     int fails = 0;
00366
00367     if (testRound()){
00368         std::cout « "testRound: Passed" « std::endl;
00369     } else {
00370         std::cout « "testRound: Failed" « std::endl;
00371         fails++;
00372     }
00373
00374     if (testCash()){
00375         std::cout « "testCash: Passed" « std::endl;
00376     } else {
00377         std::cout « "testCash: Failed" « std::endl;
00378         fails++;
00379     }
00380
00381     if (testLives()){
00382         std::cout « "testLives: Passed" « std::endl;
00383     } else {
00384         std::cout « "testLives: Failed" « std::endl;
00385         fails++;
00386     }
00387
00388     if (testGridSize()){
00389         std::cout « "testGridSize: Passed" « std::endl;
00390     } else {
00391         std::cout « "testGridSize: Failed" « std::endl;
00392         fails++;
00393     }
00394
00395     if (testGridSquareCenters()){
00396         std::cout « "testGridSquareCenters: Passed" « std::endl;
00397     } else {
00398         std::cout « "testGridSquareCenters: Failed" « std::endl;
00399         fails++;
00400     }
00401
00402     if (testCurrentRowCol()){
00403         std::cout « "testCurrentRowCol: Passed" « std::endl;
```

```
00404     } else {
00405         std::cout « "testCurrentRowCol: Failed" « std::endl;
00406         fails++;
00407     }
00408 
00409     if (testCurrentSquare()){
00410         std::cout « "testCurrentSquare: Passed" « std::endl;
00411     } else {
00412         std::cout « "testCurrentSquare: Failed" « std::endl;
00413         fails++;
00414     }
00415 
00416     if (testGetSquareByPos()){
00417         std::cout « "testGetSquareByPos: Passed" « std::endl;
00418     } else {
00419         std::cout « "testGetSquareByPos: Failed" « std::endl;
00420         fails++;
00421     }
00422 
00423     if (testNextRoad()){
00424         std::cout « "testNextRoad: Passed" « std::endl;
00425     } else {
00426         std::cout « "testNextRoad: Failed" « std::endl;
00427         fails++;
00428     }
00429 
00430     if (testFirstRoad()){
00431         std::cout « "testFirstRoad: Passed" « std::endl;
00432     } else {
00433         std::cout « "testFirstRoad: Failed" « std::endl;
00434         fails++;
00435     }
00436 
00437     if (testRead()){
00438         std::cout « "testRead: Passed" « std::endl;
00439     } else {
00440         std::cout « "testRead: Failed" « std::endl;
00441         fails++;
00442     }
00443 
00444     if (testWrite()){
00445         std::cout « "testWrite: Passed" « std::endl;
00446     } else {
00447         std::cout « "testWrite: Failed" « std::endl;
00448         fails++;
00449     }
00450 
00451     if (testAddObject()){
00452         std::cout « "testAddObject: Passed" « std::endl;
00453     } else {
00454         std::cout « "testAddObject: Failed" « std::endl;
00455         fails++;
00456     }
00457 
00458     if (TestAddObjectByType()){
00459         std::cout « "TestAddObjectByType: Passed" « std::endl;
00460     } else {
00461         std::cout « "TestAddObjectByType: Failed" « std::endl;
00462         fails++;
00463     }
00464 
00465     std::cout « "Making random map:" « std::endl;
00466     if (testRandomMap()){
00467         std::cout « "testRandom: Passed" « std::endl;
00468     } else {
00469         std::cout « "testRandom: Failed" « std::endl;
00470     }
00471 
00472     //testRandom2();
00473 
00474     if (fails == 0){
00475         std::cout « "All Level test passed" « std::endl;
00476     } else {
00477         std::cout « fails « " Level test failed" « std::endl;
00478     }
00479 
00480     return fails;
00481 }
```

9.36 ObjectTests.cpp

```
00001 #include "object.hpp"
00002 #include "level.hpp"
```

```
00003 #include "vector2d.hpp"
00004 #include <stdio.h>
00005 #include <cassert>
00006
00007 // Test for Object Constructor
00008 bool testObjectConstructor() {
00009     Level lv(1000, 1000, 50); // Create a level
00010     Vector2D pos(100, 200); // Create a position vector
00011
00012     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00013
00014     // Check if the object properties were set correctly
00015     return (obj.get_position() == pos &&
00016             obj.get_health() == 100 &&
00017             obj.get_damage() == 10 &&
00018             obj.get_range() == 50 &&
00019             obj.get_attack_speed() == 5 &&
00020             obj.get_original_attack_speed() == 5 &&
00021             obj.get_type() == 1 &&
00022             obj.get_state() == State::none);
00023 }
00024
00025 // Test for Object Getters and Setters
00026 bool testObjectGettersAndSetters() {
00027     Level lv(1000, 1000, 50); // Create a level
00028     Vector2D pos(100, 200); // Create a position vector
00029     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00030
00031     // Test Getters
00032     assert(obj.get_health() == 100);
00033     assert(obj.get_damage() == 10);
00034     assert(obj.get_range() == 50);
00035     assert(obj.get_attack_speed() == 5);
00036     assert(obj.get_original_attack_speed() == 5);
00037     assert(obj.get_position() == pos);
00038     assert(obj.get_type() == 1);
00039     assert(obj.get_attack_counter() == 0);
00040     assert(obj.get_reset_counter() == 0);
00041
00042     // Test Setters
00043     obj.set_attack_speed(6);
00044     obj.set_position(pos);
00045
00046     assert(obj.get_health() == 100);
00047     assert(obj.get_damage() == 10);
00048     assert(obj.get_range() == 50);
00049     assert(obj.get_attack_speed() == 6);
00050     assert(obj.get_position() == Vector2D(100, 200));
00051
00052     return true;
00053 }
00054
00055 // Test for Object Movement
00056 bool testObjectMovement() {
00057     Level lv(1000, 1000, 50); // Create a level
00058     Vector2D pos(100, 200); // Create a position vector
00059     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00060
00061     Vector2D new_pos(150, 250);
00062     obj.set_position(new_pos);
00063
00064     // Check if the object's position has been updated
00065     assert(obj.get_position() == new_pos);
00066
00067     return true;
00068 }
00069
00070 // Test for Object State
00071 bool testObjectState() {
00072     Level lv(1000, 1000, 50); // Create a level
00073     Vector2D pos(100, 200); // Create a position vector
00074     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00075
00076     obj.set_state(State::attacking_left); // Set object state to ATTACK
00077
00078     // Check if the object's state has been updated
00079     assert(obj.get_state() == State::attacking_left);
00080
00081     return true;
00082 }
00083
00084 // Test for Object Distance Calculation
00085 bool testObjectDistanceCalculation() {
00086     Level lv(1000, 1000, 50); // Create a level
00087     Vector2D pos(100, 200); // Create a position vector
00088     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00089
```

```
00090     Vector2D target_pos(150, 250);
00091
00092     // Calculate distance between object and target position
00093     double distance = obj.distance_to(target_pos);
00094
00095     return true;
00096 }
00097
00098 // Test for Object Health Management
00099 bool testObjectHealthManagement() {
00100     Level lv(1000, 1000, 50); // Create a level
00101     Vector2D pos(100, 200); // Create a position vector
00102     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00103
00104     obj.lose_health(20); // Lose health
00105
00106     // Check if the object's health decreased correctly
00107     assert(obj.get_health() == 80);
00108
00109     obj.gain_health(30); // Gain health
00110
00111     // Check if the object's health increased correctly
00112     assert(obj.get_health() == 110);
00113
00114     return true;
00115 }
00116
00117 // Test for Object Attack Speed Management
00118 bool testObjectAttackSpeedManagement() {
00119     Level lv(1000, 1000, 50); // Create a level
00120     Vector2D pos(100, 200); // Create a position vector
00121     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00122
00123     obj.gain_attack_speed(2); // Gain attack speed
00124
00125     // Check if the object's attack speed decreased correctly
00126     assert(obj.get_attack_speed() == 3);
00127
00128     obj.set_attack_speed(obj.get_original_attack_speed());
00129     obj.lose_attack_speed(1); // Lose attack speed
00130
00131     // Check if the object's attack speed increased correctly
00132     assert(obj.get_attack_speed() == 6);
00133
00134     return true;
00135 }
00136
00137 // Test for Object Counter Management
00138 bool testObjectCounterManagement() {
00139     Level lv(1000, 1000, 50); // Create a level
00140     Vector2D pos(100, 200); // Create a position vector
00141     Object obj(lv, pos, 100, 10, 50, 5, 1); // Create an object
00142
00143     obj.set_attack_counter(2); // Set attack counter
00144     obj.set_reset_counter(3); // Set reset counter
00145
00146     obj.attack_counter_up(); // Increment attack counter
00147     obj.reset_counter_up(); // Increment reset counter
00148
00149     // Check if the counters incremented correctly
00150     assert(obj.get_attack_counter() == 1);
00151     assert(obj.get_reset_counter() == 4);
00152
00153     return true;
00154 }
00155
00156 int object_tests() {
00157     int failed_tests = 0;
00158
00159     if (!testObjectConstructor()) {
00160         std::cout « "testObjectConstructor failed!" « std::endl;
00161         failed_tests++;
00162     } else {
00163         std::cout « "testObjectConstructor passed!" « std::endl;
00164     }
00165
00166     if (!testObjectGettersAndSetters()) {
00167         std::cout « "testObjectGettersAndSetters failed!" « std::endl;
00168         failed_tests++;
00169     } else {
00170         std::cout « "testObjectGettersAndSetters passed!" « std::endl;
00171     }
00172
00173     if (!testObjectMovement()) {
00174         std::cout « "testObjectMovement failed!" « std::endl;
00175         failed_tests++;
00176     } else {
```

```
00177         std::cout « "testObjectMovement passed!" « std::endl;
00178     }
00179
00180     if (!testObjectState()) {
00181         std::cout « "testObjectState failed!" « std::endl;
00182         failed_tests++;
00183     } else {
00184         std::cout « "testObjectState passed!" « std::endl;
00185     }
00186
00187     if (!testObjectDistanceCalculation()) {
00188         std::cout « "testObjectDistanceCalculation failed!" « std::endl;
00189         failed_tests++;
00190     } else {
00191         std::cout « "testObjectDistanceCalculation passed!" « std::endl;
00192     }
00193
00194     if (!testObjectHealthManagement()) {
00195         std::cout « "testObjectHealthManagement failed!" « std::endl;
00196         failed_tests++;
00197     } else {
00198         std::cout « "testObjectHealthManagement passed!" « std::endl;
00199     }
00200
00201     if (!testObjectAttackSpeedManagement()) {
00202         std::cout « "testObjectAttackSpeedManagement failed!" « std::endl;
00203         failed_tests++;
00204     } else {
00205         std::cout « "testObjectAttackSpeedManagement passed!" « std::endl;
00206     }
00207
00208     if (!testObjectCounterManagement()) {
00209         std::cout « "testObjectCounterManagement failed!" « std::endl;
00210         failed_tests++;
00211     } else {
00212         std::cout « "testObjectCounterManagement passed!" « std::endl;
00213     }
00214
00215     if (failed_tests == 0) {
00216         std::cout « "All Object tests passed!" « std::endl;
00217     } else {
00218         std::cout « failed_tests « " Object tests failed!" « std::endl;
00219     }
00220
00221     return failed_tests;
00222 }
```

9.37 SquareTests.cpp

```
00001 #include "square.hpp"
00002 #include <iostream>
00003
00004 bool testCenter(){
00005     Vector2D cent(2, 3);
00006     Square sq(cent);
00007     return sq.get_center() == cent;
00008 }
00009
00010 bool testOccupied(){
00011     Vector2D cent(2, 3), cent2(4, 5), cent3(6, 7);
00012     Square sq(cent), sq2(cent2), sq3(cent3);
00013     sq.occupy_by_grass();
00014     sq2.occupy_by_road();
00015     sq3.occupy_by_tower();
00016     return sq.get_occupied() == grass && sq2.get_occupied() == road && sq3.get_occupied() == tower;
00017 }
00018
00019 static int square_test() {
00020     int fails = 0;
00021
00022     if (testCenter()){
00023         std::cout « "testCenter: Passed" « std::endl;
00024     } else {
00025         std::cout « "testCenter: Failed" « std::endl;
00026         fails++;
00027     }
00028
00029     if (testOccupied()){
00030         std::cout « "testOccupied: Passed" « std::endl;
00031     } else {
00032         std::cout « "testOccupied: Failed" « std::endl;
00033         fails++;
00034     }
```

```
00035
00036     if (fails == 0){
00037         std::cout « "All Square test passed" « std::endl;
00038     } else {
00039         std::cout « fails « " Square test failed" « std::endl;
00040     }
00041
00042     return fails;
00043 }
```