The Remaining of Us

Alpha

Generated by Doxygen 1.9.7

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

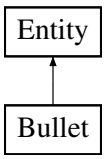
Chapter 4

Class Documentation

4.1 Bullet Class Reference

`#include <Bullet.h>`

Inheritance diagram for Bullet:



Public Member Functions

- Bullet (sf::Vector2f pos, sf::Vector2f size, int sht_timer)
- ~**Bullet** ()

  Bullet destructor.
- sf::RectangleShape **getShape** ()

  Getter used for getting bullet's shape.
- void resetPos (sf::Vector2f pos)
- bool **getIsShot** ()

  Getter which returns isShot value.
- void **isShotTrue** ()

  Sets isShot value to True.
- void **isShotFalse** ()

  Sets isShot value to False.
- int **getShotTimer** ()

  Getter which returns shot timer.

Public Member Functions inherited from Entity

- **Entity** ()

  Default Entity constructor.
- Entity (int lives_ct, sf::Vector2f pos)
- virtual ~**Entity** ()

  Entity destructor.
- void setTexture (std::string path)
- void loadSpritesFromSheet (int spriteWidth, int spriteHeight, int numRows, int numColumns)
- void setSprite (int num_of_sprite, sf::Vector2f pos)
- void Draw (sf::RenderWindow &target)
- void **Animation** ()

  Runs entity's animation.
- sf::Vector2f & **getPos** ()

  Getter used for getting position.
- sf::Vector2f **getCenter** ()

  Getter used for getting center of the entity.
- sf::Sprite & **getSprite** ()

  Getter used for getting entity's current sprite.
- int & **getLives** ()

  Getter used for getting lives count.
- void **livesDecr** ()

  Decrements lives count.
- void setPos (sf::Vector2f pos)
- void setAnimationFrequency (int freq)
- void **updateHealthBar** ()

  Refreshes healthbar, based on current lives count : max lives count ratio.

4.1.1 Detailed Description

Class which defines bullet objects

Entity is it's parent class

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Bullet()

```
Bullet::Bullet (
            sf::Vector2f pos,
            sf::Vector2f size,
            int sht_timer )
```

Bullet multi-argument constructor

Parameters

pos	Bullet's initial position
size	Size of the bullet
sht_timer	shootTimer value

4.1.3 Member Function Documentation

4.1.3.1 resetPos()

```
void Bullet::resetPos (
            sf::Vector2f pos )
```

Resets position to given value

Parameters

pos	New position

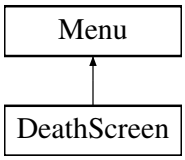
The documentation for this class was generated from the following files:

- Bullet.h
- Bullet.cpp

4.2 DeathScreen Class Reference

```
#include <DeathScreen.h>
```

Inheritance diagram for DeathScreen:



Public Member Functions

- DeathScreen (int k_time, std::string f_path, std::filesystem::path backgrd_path)
- void setValues (sf::RenderWindow &window, int val)
- void set2PlayerValues (sf::RenderWindow &window, int val, int which_won)
- void moveDown ()

  Polimorphic moveDown method.

Public Member Functions inherited from Menu

- Menu (int k_time, std::string f_path, std::filesystem::path backgrd_path)
- ~**Menu** ()

  Menu destructor.
- void draw (sf::RenderWindow &window)
- virtual void **setValues** ()

  Initiates menu's values.
- void moveUp (int min_opt)
- virtual void moveDown ()

Used to switch to lower option.

- int & **getKeytime** ()

  Getter which returns keytime.

- int **getSelectedItem** ()

  Getter which returns selected item.

- sf::Font & **getFont** ()

  Getter which returns Font.

- void **setText** (int i, sf::Text text)

  Setter which sets text value to the adequate Text by its index i.

- sf::Sprite & **getBackground** ()

  Getter which returns background sprite.

- int & **getSelectedItemIndex** ()

  Getter which returns selected item index.

- sf::Text & **getMenuText** (int i)

  Getter which returns option's text by it's id i.

4.2.1 Detailed Description

A Menu derived class which acts like a menu but is used as a deathscreen

It could also be called Deathscreen Menu

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DeathScreen()

```
DeathScreen::DeathScreen (
            int k_time,
            std::string f_path,
            std::filesystem::path backgrd_path )
```

Multi-argument constructor

Parameters

k_time	Keytime value
f_path	Font file's directory
backgrd_path	Background png's directory

4.2.3 Member Function Documentation

4.2.3.1 moveDown()

```
void DeathScreen::moveDown ( )  [virtual]
```

Polimorphic moveDown method.

Reimplemented from Menu.

4.2.3.2 set2PlayerValues()

```
void DeathScreen::set2PlayerValues (
            sf::RenderWindow & window,
            int val,
            int which_won )
```

This method initializes deathscreen for two-player mode

Parameters

window	Target window
val	Score
which_won	States which player won the game (had a higher score/draw also possible)

4.2.3.3 setValues()

```
void DeathScreen::setValues (
            sf::RenderWindow & window,
            int val )
```

This method initializes the deathscreen

Parameters

window	Target window
val	Score

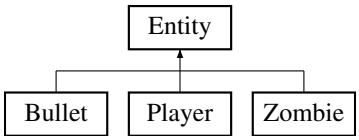
The documentation for this class was generated from the following files:

- DeathScreen.h
- DeathScreen.cpp

4.3 Entity Class Reference

```
#include <Entity.h>
```

Inheritance diagram for Entity:



Public Member Functions

- **Entity** ()

  Default Entity constructor.
- Entity (int lives_ct, sf::Vector2f pos)
- virtual ~**Entity** ()

  Entity destructor.
- void setTexture (std::string path)
- void loadSpritesFromSheet (int spriteWidth, int spriteHeight, int numRows, int numColumns)
- void setSprite (int num_of_sprite, sf::Vector2f pos)
- void Draw (sf::RenderWindow &target)
- void **Animation** ()

  Runs entity's animation.
- sf::Vector2f & **getPos** ()

  Getter used for getting position.
- sf::Vector2f **getCenter** ()

  Getter used for getting center of the entity.
- sf::Sprite & **getSprite** ()

  Getter used for getting entity's current sprite.
- int & **getLives** ()

  Getter used for getting lives count.
- void **livesDecr** ()

  Decrements lives count.
- void setPos (sf::Vector2f pos)
- void setAnimationFrequency (int freq)
- void **updateHealthBar** ()

  Refreshes healthbar, based on current lives count : max lives count ratio.

4.3.1 Detailed Description

Entity is an abstract class which defines basic attributes and methods for players, npcs and more

E.g. in this program it is used for Player, Zombie and bullet

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Entity()

```
Entity::Entity (
            int lives_ct,
            sf::Vector2f pos )
```

Entity multi-argument constructor

Used for setting starting values

Parameters

lives_ct	Indicates how many lives an entity should start with
pos	Indicates what starting postition should an entity have

4.3.3 Member Function Documentation

4.3.3.1 Draw()

```
void Entity::Draw (
            sf::RenderWindow & target )
```

Draws entity's sprite to a target window

Parameters

target	Target window

4.3.3.2 loadSpritesFromSheet()

```
void Entity::loadSpritesFromSheet (
            int spriteWidth,
            int spriteHeight,
            int numRows,
            int numColumns )
```

Loads sprites from sheet and stores them in the vector

Sprites are loaded from a png file which consists of all the needed entity sprites

Parameters

spriteWidth	Width of one sprite (in pixels)
spriteHeight	Height of the whole png
numRows	Number of rows
numColumns	Number of columns

4.3.3.3 setAnimationFrequency()

```
void Entity::setAnimationFrequency (
            int freq )
```

Setter used for setting animation frequency

Parameters

freq	Frequency of sprites switches

4.3.3.4 setPos()

```
void Entity::setPos (
            sf::Vector2f pos )
```

Sets position to a new one

Parameters

pos	New position

4.3.3.5 setSprite()

```
void Entity::setSprite (
            int num_of_sprite,
            sf::Vector2f pos )
```

Used for setting a sprite and position

Parameters

num_of_sprite	Index of wanted sprite
pos	Wanted position

4.3.3.6 setTexture()

```
void Entity::setTexture (
            std::string path )
```

Setter used for setting texture attribute value

Parameters

path	Path of an input file

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

- Entity.h
- Entity.cpp

4.4 Game Class Reference

```
#include <Game.h>
```

Public Member Functions

- **Game** ()

  Game default constructor.
- ∼**Game** ()

  Game destructor.

- void **run** ()

  Run method is used to play the game in main function. This method is all that's needed for the game to run in main function.
- void Update (Player &player1, Player &player2, Zombie &basiczombie, Zombie &strong_zombie, Zombie &fast_zombie, Bullet &bullet, sf::Clock &reloadClock, sf::Clock &reloadClock2, HUD &hud, HUD &hud2, Menu &menu, DeathScreen &deathscreen, DeathScreen &duo_ds, int &option)
- void processEvents (Zombie &zombie, Zombie &strong_zombie, Zombie &fast_zombie)
- void zombieHandling (Player &player1, Zombie &zombie, Zombie &strong_zombie, Zombie &fast_zombie, HUD &hud)
- void zombieHandling_2Player (Player &player1, Player &player2, Zombie &zombie, Zombie &strong_zombie, Zombie &fast_zombie, HUD &hud, HUD &hud2)
- void loadPairsFromFile (std::filesystem::path path)
- void drawPNHPairs (sf::RenderWindow &target_window)

4.4.1 Detailed Description

This class is responsible for running the game loop

Game class uses all the created classes to process the game and run it using the run() method

4.4.2 Member Function Documentation

4.4.2.1 drawPNHPairs()

```
void Game::drawPNHPairs (
            sf::RenderWindow & target_window )
```

This method draw leaderboard file pairs of players and their highscores to a target window

Parameters

target_window	Window to which pairs are drawn

4.4.2.2 loadPairsFromFile()

```
void Game::loadPairsFromFile (
            std::filesystem::path path )
```

This method is used for loading player_n_highscore pairs from a leaderboard file

Parameters

path	Path parameter containing input file's location

4.4.2.3 processEvents()

```
void Game::processEvents (
```

```
            Zombie & zombie,
            Zombie & strong_zombie,
            Zombie & fast_zombie )
```

This method is used for processing different events

It processes events like: player nickname input or closing the game window with mouse

Parameters

zombie	Basic zombie type object. It's resetting methods are needed e.g. to reset spawn clocks before the game is initiated
strong_zombie	Strong zombie type object. It's resetting methods are needed e.g. to reset spawn clocks before the game is initiated
fast_zombie	Fast zombie type object. It's resetting methods are needed e.g. to reset spawn clocks before the game is initiated

4.4.2.4 Update()

```
void Game::Update (
            Player & player1,
            Player & player2,
            Zombie & basiczombie,
            Zombie & strong_zombie,
            Zombie & fast_zombie,
            Bullet & bullet,
            sf::Clock & reloadClock,
            sf::Clock & reloadClock2,
            HUD & hud,
            HUD & hud2,
            Menu & menu,
            DeathScreen & deathscreen,
            DeathScreen & duo_ds,
            int & option )
```

Update method is used for drawing all the video on screen as well as for the whole game mechanism

Update method draws all the necessary objects in a form of sprites or sf::Text objects

Parameters

player1	Player object used for singleplayer and for Player no 1 in two-player mode
player2	Player object used for player no 2 in two-player mode
basiczombie	Zombie class object used for creating basic-type zombies
strong_zombie	Zombie class object used for creating strong-type zombies
fast_zombie	Zombie class object used for creating fast-type zombies
bullet	Bullet class object used for creating bullets in game that are later used by Player to shoot
reloadClock	Variable used for timed sprite switches during reload animation
reloadClock2	The same as reloadClock but for second player
hud	HUD object used for drawing head-up display on screen
hud2	The same as hud but for second player (used in two-player mode)
menu	Object used for Main Menu handling in game
deathscreen	Used for displaying game's deathscreen after the player dies
duo_ds	Just like deathscreen but for two-player mode
option	Used for defining which case is used, adequate to the Game class opt attribute

4.4.2.5 zombieHandling()

```
void Game::zombieHandling (
            Player & player1,
            Zombie & zombie,
            Zombie & strong_zombie,
            Zombie & fast_zombie,
            HUD & hud )
```

Method used for spawning zombies

Its used for handling zombies' behaviour (spawning, collisions etc.) + some other aspects like updating hud's score count when a zombie is killed

Parameters

player1	Player class object used to decrease health when zombie touches the bottom
zombie	Zombie class object for basic-type zombies
strong_zombie	Zombie class object for strong-type zombies
fast_zombie	Zombie class object for fast-type zombies
hud	HUD object needed for updating HUD.

4.4.2.6 zombieHandling_2Player()

```
void Game::zombieHandling_2Player (
            Player & player1,
            Player & player2,
            Zombie & zombie,
            Zombie & strong_zombie,
            Zombie & fast_zombie,
            HUD & hud,
            HUD & hud2 )
```

Slightly altered version of zombieHandling method

It also needs second player's object and second hud to work in two-player mode

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

- Game.h
- Game.cpp

4.5 HUD Class Reference

```
#include <HUD.h>
```

Public Member Functions

- **HUD** ()

  HUD default constructor.
- HUD (int score_start, int max_ammo_start, std::string player_name, std::string font_path)
- ~**HUD** ()

  HUD destructor.
- void drawHUD (sf::RenderWindow &target_window)
- void updateScore (int added_ammount)
- void changeMaxAmmo (int new_ammo)
- void setHUDfor2ndPlayer (sf::Vector2f offset)
- void changeCurrAmmo (int add_to_ammo)
- int & **getScoreCountInt** ()

  Getter used for getting score as an int.
- void **resetHUD** ()

  Resets hud's values.

4.5.1 Detailed Description

This class is used for creating and displaying heads-up-display

HUD can display current ammo, max ammo, score etc.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 HUD()

```
HUD::HUD (
            int score_start,
            int max_ammo_start,
            std::string player_name,
            std::string font_path )
```

Multi-argument constructor

Parameters

score_start	Initial score int
max_ammo_start	Initial max ammo
player_name	Player's name
font_path	Directory of the font file

4.5.3 Member Function Documentation

4.5.3.1 changeCurrAmmo()

```
void HUD::changeCurrAmmo (
            int add_to_ammo )
```

This method changes current ammo

Parameters

add_to_ammo	Ammount added to ammo

4.5.3.2 changeMaxAmmo()

```
void HUD::changeMaxAmmo (
            int new_ammo )
```

Changes max ammo value

Parameters

new_ammo	New ammo value

4.5.3.3 drawHUD()

```
void HUD::drawHUD (
            sf::RenderWindow & target_window )
```

This method draws HUD on screen

Parameters

target_window	Target window

4.5.3.4 setHUDfor2ndPlayer()

```
void HUD::setHUDfor2ndPlayer (
            sf::Vector2f offset )
```

Adapts the hud for 2nd player

Parameters

offset	Vector of floats which indicates how the hud should be moved from it's initial position

4.5.3.5 updateScore()

```
void HUD::updateScore (
            int added_ammount )
```

Used to update score in HUD

Parameters

added_ammount	Ammount added to the score

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

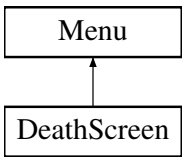
- HUD.h
- HUD.cpp

4.6 Menu Class Reference

This class dictates the behaviour of main menu.

`#include <Menu.h>`

Inheritance diagram for Menu:



Public Member Functions

- Menu (int k_time, std::string f_path, std::filesystem::path backgrd_path)
- ~**Menu** ()

  Menu destructor.
- void draw (sf::RenderWindow &window)
- virtual void **setValues** ()

  Initiates menu's values.
- void moveUp (int min_opt)
- virtual void moveDown ()

  Used to switch to lower option.
- int & **getKeytime** ()

  Getter which returns keytime.
- int **getSelectedItem** ()

  Getter which returns selected item.
- sf::Font & **getFont** ()

  Getter which returns Font.
- void **setText** (int i, sf::Text text)

  Setter which sets text value to the adequate Text by its index i.
- sf::Sprite & **getBackground** ()

  Getter which returns background sprite.
- int & **getSelectedItemIndex** ()

  Getter which returns selected item index.
- sf::Text & **getMenuText** (int i)

  Getter which returns option's text by it's id i.

4.6.1 Detailed Description

This class dictates the behaviour of main menu.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Menu()

```
Menu::Menu (
            int k_time,
            std::string f_path,
            std::filesystem::path backgrd_path )
```

Menu multi-argument constructor

Parameters

k_time	Sets key time value
f_path	Font file directory
backgrd_path	Directory of background png

4.6.3 Member Function Documentation

4.6.3.1 draw()

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

This method draws the menu on screen

It is used to draw all the sf::Text variables

Parameters

window	Target window

4.6.3.2 moveDown()

```
void Menu::moveDown ( )  [virtual]
```

Used to switch to lower option.

Reimplemented in DeathScreen.

4.6.3.3 moveUp()

```
void Menu::moveUp (
            int min_opt )
```

Used to switch to a higher option

Parameters

min_opt	Dictates which sf::Text variable is the top one

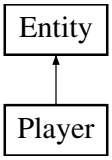
The documentation for this class was generated from the following files:

- Menu.h
- Menu.cpp

4.7 Player Class Reference

`#include <Player.h>`

Inheritance diagram for Player:



Public Member Functions

- **Player** ()
  Player default constructor.
- Player (int lives_ct, int max_amm, int curr_amm, sf::Vector2f pos)
- ~**Player** ()
  Player destructor.
- void **resetAmmo** (HUD &hud)
  Resets ammo (Also on HUD)
- void player1Movement (int windsize)
- void player2Movement (int windsize)
- bool playerReload (sf::Clock &reloadClock, sf::Keyboard::Key key)
- bool shootPistol (sf::Keyboard::Key shootkey, Bullet &bullet, sf::RenderWindow &target_window, sf::Clock &reloadClock, HUD &hud)
- int **getBulletClipSize** ()
  Getter used for getting size of the bullet clip.
- std::vector< sf::RectangleShape > & **getBulletClip** ()
  Getter which returns bullet clip.
- int **getCurrAmmo** ()
  Getter which returns current ammo value as an integer.
- int **getMaxAmmo** ()
  Getter which returns max ammo value as an integer.
- bool **getIsShooting** ()
  Getter which returns isShooting value.
- bool **getIsReloading** ()
  Getter which returns isReloading value.
- void changeIsShootingValue (bool val)
- void **playerReset** ()
  This method resets all player's attributes to their beginning state.
- void shootingAnimation (sf::Clock reloadclock)
- void reloadAnimation (sf::Clock reloadclock, HUD &hud)
- void Draw (sf::RenderWindow &target)
- int **getShotTimer** ()
  Getter used for getting shotTimer.

Public Member Functions inherited from Entity

- **Entity** ()

  Default Entity constructor.
- Entity (int lives_ct, sf::Vector2f pos)
- virtual ~**Entity** ()

  Entity destructor.
- void setTexture (std::string path)
- void loadSpritesFromSheet (int spriteWidth, int spriteHeight, int numRows, int numColumns)
- void setSprite (int num_of_sprite, sf::Vector2f pos)
- void Draw (sf::RenderWindow &target)
- void **Animation** ()

  Runs entity's animation.
- sf::Vector2f & **getPos** ()

  Getter used for getting position.
- sf::Vector2f **getCenter** ()

  Getter used for getting center of the entity.
- sf::Sprite & **getSprite** ()

  Getter used for getting entity's current sprite.
- int & **getLives** ()

  Getter used for getting lives count.
- void **livesDecr** ()

  Decrements lives count.
- void setPos (sf::Vector2f pos)
- void setAnimationFrequency (int freq)
- void **updateHealthBar** ()

  Refreshes healthbar, based on current lives count : max lives count ratio.

4.7.1 Detailed Description

Player class defines players' objects

Player class defines all the attributes and methods needed for an Entity class to transform into player

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Player()

```
Player::Player (
            int lives_ct,
            int max_amm,
            int curr_amm,
            sf::Vector2f pos )
```

Player multi-argument constructor

Parameters

lives_ct	Initial lives count
max_amm	Maximum ammo initial value
curr_amm	Current ammo initial value
pos	Initial position

4.7.3 Member Function Documentation

4.7.3.1 changeIsShootingValue()

```
void Player::changeIsShootingValue (
            bool val )
```

isShooting setter

Parameters

val	IsShooting's new value

4.7.3.2 Draw()

```
void Player::Draw (
            sf::RenderWindow & target )
```

Polimorphic Draw method used to properly draw player to target window

Parameters

target	Target window

4.7.3.3 player1Movement()

```
void Player::player1Movement (
            int windsize )
```

Defines how the first player moves

Parameters

windsize	size of window

4.7.3.4 player2Movement()

```
void Player::player2Movement (
            int windsize )
```

Defines how the second player moves

Parameters

windsize	size of window

4.7.3.5 playerReload()

```
bool Player::playerReload (
            sf::Clock & reloadClock,
            sf::Keyboard::Key key )
```

Performs the reloading procedure and returns true if a reload was initiated

Parameters

reloadClock	Used for counting the duration of reload
key	Passed key variable defines which key is then used for reloading

4.7.3.6 reloadAnimation()

```
void Player::reloadAnimation (
            sf::Clock reloadclock,
            HUD & hud )
```

Used to perform player's reloading animation

Parameters

reloadclock	Times the reload animation
hud	Updates hud when finished

4.7.3.7 shootingAnimation()

```
void Player::shootingAnimation (
            sf::Clock reloadclock )
```

Used to perform player's shooting animation

Parameters

reloadclock	Times the animatioon

4.7.3.8 shootPistol()

```
bool Player::shootPistol (
            sf::Keyboard::Key shootkey,
            Bullet & bullet,
            sf::RenderWindow & target_window,
            sf::Clock & reloadClock,
            HUD & hud )
```

This method is used for player's shooting and returns true if shot was initiated

While used in the game_loop, player can shoot bullets when a given key is pressed

Parameters

shootkey	Key used for shooting
bullet	Bullet objects used for drawing them
target_window	Target window for drawing
reloadClock	Used for timing the reload
hud	HUD is needed so that the ammo count on screen can be updated

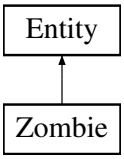
The documentation for this class was generated from the following files:

- Player.h
- Player.cpp

4.8 Zombie Class Reference

`#include <Zombie.h>`

Inheritance diagram for Zombie:



Public Member Functions

- **Zombie** ()

  Zombie default constructor.
- Zombie (int score_val, int lives_ct, int spawn_freq_ms, sf::Vector2f pos, float mov_spd)
- ~**Zombie** ()

  Zombie Destructor.
- float **getMovSpeed** ()

  Getter used for getting movement speed.
- int **getCurrSpawnFreq** ()

  Getter used for getting current spawn frequency.
- void **setCurrSpawnFreq** (int val)

  Setter used for setting current spawn frequency.
- int **getBaseSpawnFreq** ()

  Getter used for getting Base spawn frequency.
- void **setBaseSpawnFreq** (int val)

  Setter used for setting Base spawn frequency.
- void **setCurrMovSpeed** (int val)

  setter used for setting current movement speed
- void **resetValues** ()

  Resets all Zombie attributes to their initial state.
- void **restartClock** ()

  Restarts spawn clock.
- sf::Clock & **getSpawnClock** ()

  Getter used for getting spawn clock.
- int **getScoreValue** ()

  Getter used for getting score value.

Public Member Functions inherited from Entity

- **Entity** ()

  Default Entity constructor.
- Entity (int lives_ct, sf::Vector2f pos)
- virtual ~**Entity** ()

  Entity destructor.
- void setTexture (std::string path)
- void loadSpritesFromSheet (int spriteWidth, int spriteHeight, int numRows, int numColumns)
- void setSprite (int num_of_sprite, sf::Vector2f pos)
- void Draw (sf::RenderWindow &target)
- void **Animation** ()

  Runs entity's animation.
- sf::Vector2f & **getPos** ()

  Getter used for getting position.
- sf::Vector2f **getCenter** ()

  Getter used for getting center of the entity.
- sf::Sprite & **getSprite** ()

  Getter used for getting entity's current sprite.
- int & **getLives** ()

  Getter used for getting lives count.
- void **livesDecr** ()

  Decrements lives count.
- void setPos (sf::Vector2f pos)
- void setAnimationFrequency (int freq)
- void **updateHealthBar** ()

  Refreshes healthbar, based on current lives count : max lives count ratio.

4.8.1 Detailed Description

This class defines zombie objects which are enemies in this game

Different types of zombies can be created with this class. It also inherits from the Entity abstract class

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Zombie()

```
Zombie::Zombie (
            int score_val,
            int lives_ct,
            int spawn_freq_ms,
            sf::Vector2f pos,
            float mov_spd )
```

Multi argument constructor

Parameters

score_val	Amount of score gained after killing
lives_ct	No of lives
spawn_freq_ms	Spawn frequency in ms
pos	Initial position
mov_spd	Initial movement speed

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

- Zombie.h
- Zombie.cpp

Chapter 5

File Documentation

5.1 Bullet.h

```
00001 #pragma once
00002 #include "Entity.h"
00003
00007 class Bullet : public Entity
00008 {
00009 private:
00011     sf::RectangleShape bullet_shape;
00013     bool isShot;
00015     int shootTimer;
00016 public:
00022     Bullet(sf::Vector2f pos, sf::Vector2f size, int sht_timer);
00024     ~Bullet();
00026     sf::RectangleShape getShape();
00030     void resetPos(sf::Vector2f pos);
00032     bool getIsShot();
00034     void isShotTrue();
00036     void isShotFalse();
00038     int getShotTimer();
00039 };
00040
```

5.2 DeathScreen.h

```
00001 #pragma once
00002 #include "Menu.h"
00003
00007 class DeathScreen : public Menu
00008 {
00010     sf::Text dstext[4];
00012     int score_ct_int;
00013 public:
00019     DeathScreen(int k_time, std::string f_path, std::filesystem::path backgrd_path);
00024     void setValues(sf::RenderWindow &window, int val);
00030     void set2PlayerValues(sf::RenderWindow& window, int val, int which_won);
00032     void moveDown();
00033 };
```

5.3 Entity.h

```
00001 #pragma once
00002 #include <iostream>
00003 #include <SFML/Graphics.hpp>
00004 #include <chrono>
00005 #include <string>
00006 #include <vector>
00007 #include <random>
00008 #include <filesystem>
00009
00013 class Entity
```

```
00014 {
00015 private:
00017     int lives;
00019     int max_lives;
00021     int spriteCount = 0;
00023     sf::Texture texture;
00025     sf::Sprite sprite;
00027     std::vector<sf::Sprite> spriteVector;
00029     sf::Vector2f position;
00031     sf::Clock animation_clock;
00033     int currSprite = 0;
00035     int freq_of_animation_switch = 0;
00037     sf::RectangleShape health_bar;
00039     float health_bar_offset;
00040
00041 public:
00043     Entity();
00049     Entity(int lives_ct, sf::Vector2f pos);
00051     virtual ~Entity();
00055     void setTexture(std::string path);
00063     void loadSpritesFromSheet(int spriteWidth, int spriteHeight, int numRows, int numColumns);
00068     void setSprite(int num_of_sprite, sf::Vector2f pos);
00072     void Draw(sf::RenderWindow& target);
00074     void Animation();
00076     sf::Vector2f& getPos();
00078     sf::Vector2f getCenter();
00080     sf::Sprite& getSprite();
00082     int &getLives();
00084     void livesDecr();
00088     void setPos(sf::Vector2f pos);
00092     void setAnimationFrequency(int freq);
00094     void updateHealthBar();
00095 };
```

5.4 Functions.h

```
00001 #pragma once
00002 #include "SFML\Audio.hpp"
00003 #include <filesystem>
00004 #include <ranges>
00005 #include <fstream>
00006 #include <iostream>
00007 #include "Zombie.h"
00008 #include <string>
00009 #include <future>
00010
00014 std::vector<std::string> readLinesFromFile(const std::string& filename);
00015
00021 void updateLeaderboardFile(std::filesystem::path leaderboard_path, std::string player_nick, int
      score);
00022
00029 void increaseDifficulty(Zombie& zombie, Zombie& strong_zombie, HUD& hud);
00030
00035 void loadAudioPromise(std::promise<sf::SoundBuffer> promise, std::string file_path);
```

5.5 Game.h

```
00001 #pragma once
00002
00003
00004 #include <iostream>
00005 #include <fstream>
00006 #include <string>
00007 #include <filesystem>
00008 #include <regex>
00009 #include <ranges>
00010 #include <algorithm>
00011 #include <tuple>
00012 #include <cmath>
00013 #include <chrono>
00014 #include <future>
00015
00016 //Custom classes
00017 #include "Player.h"
00018 #include "Menu.h"
00019 #include "HUD.h"
00020 #include "Zombie.h"
00021 #include "DeathScreen.h"
00022 #include "Functions.h"
```

```
00023
00024 //SFML libraries
00025 #include "SFML\System.hpp"
00026 #include "SFML\Audio.hpp"
00027 #include "SFML\Graphics.hpp"
00028 #include "SFML\Network.hpp"
00029 #include "SFML\Window.hpp"
00030
00031
00032
00036 class Game
00037 {
00038 private:
00040     int opt = -4;
00042     sf::RenderWindow window;
00044     sf::Texture texture;
00046     sf::Texture intro_texture;
00048     sf::Texture outro_texture;
00050     sf::Texture instruct_texture;
00052     sf::Sprite background;
00054     sf::Sprite intro;
00056     sf::Sprite outro;
00058     sf::Sprite instruct;
00060     std::vector<Zombie> vec_of_zombies;
00061
00063     sf::Text player_name_render;
00065     std::vector<sf::Text> vec_of_leaderboard_texts;
00067     sf::Text nickInputPrompt;
00069     sf::Font font;
00071     std::string player_name;
00072
00074     std::tuple<int, std::string> player_n_highscore;
00076     std::vector<std::tuple<int, std::string» vec_of_pairs;
00078     std::filesystem::path leaderboard_file;
00080     std::regex input_regex;
00081
00083     sf::SoundBuffer intro_buffer;
00085     sf::SoundBuffer game_buffer;
00087     sf::SoundBuffer deathscreen_buffer;
00089     sf::SoundBuffer zombie_buffer;
00091     sf::SoundBuffer shot_buffer;
00093     sf::SoundBuffer reload_buffer;
00094
00096     sf::Sound intro_sound;
00098     sf::Sound game_sound;
00100     sf::Sound deathscreen_sound;
00102     sf::Sound zombie_sound[24];
00104     sf::Sound shot_sound[24];
00106     sf::Sound reload_sound[24];
00108     sf::Clock intro_outro_clk;
00109
00110 public:
00112     Game();
00114     ~Game();
00116     void run();
00134     void Update(Player& player1, Player& player2, Zombie& basiczombie, Zombie& strong_zombie, Zombie&
      fast_zombie, Bullet& bullet,
00135         sf::Clock& reloadClock, sf::Clock& reloadClock2, HUD& hud, HUD &hud2, Menu& menu, DeathScreen&
      deathscreen, DeathScreen &duo_ds, int& option);
00142     void processEvents(Zombie& zombie, Zombie& strong_zombie, Zombie& fast_zombie);
00151     void zombieHandling(Player& player1, Zombie& zombie, Zombie& strong_zombie, Zombie& fast_zombie,
      HUD& hud);
00155     void zombieHandling_2Player(Player& player1, Player& player2, Zombie& zombie, Zombie&
      strong_zombie, Zombie& fast_zombie, HUD& hud, HUD& hud2);
00159     void loadPairsFromFile(std::filesystem::path path);
00163     void drawPNHPairs(sf::RenderWindow& target_window);
00164 };
```

5.6 HUD.h

```
00001 #pragma once
00002 #include "Entity.h"
00003 #include <sstream>
00004
00008 class HUD
00009 {
00010 private:
00012     int score_count_integer;
00014     int curr_ammo;
00016     int max_ammo;
00018     int max_ammo_begin;
00019
00021     sf::Font font;
```

```
00023     sf::Text score_count;
00025     sf::Text max_ammo_text;
00027     sf::Text curr_ammo_text;
00029     sf::Text ammo_word_text;
00031     sf::Text score_text;
00032
00034     std::string player_name;
00035 public:
00037     HUD();
00044     HUD(int score_start, int max_ammo_start, std::string player_name, std::string font_path);
00046     ~HUD();
00050     void drawHUD(sf::RenderWindow &target_window);
00054     void updateScore(int added_ammount);
00058     void changeMaxAmmo(int new_ammo);
00062     void setHUDfor2ndPlayer(sf::Vector2f offset);
00066     void changeCurrAmmo(int add_to_ammo);
00068     int &getScoreCountInt();
00070     void resetHUD();
00071 };
00072
```

5.7 Menu.h

```
00001 #pragma once
00002 #include "Entity.h"
00003
00005 class Menu
00006 {
00007 private:
00009     int keytime;
00011     std::string font_path;
00013     int selected_item_index = 0;
00015     sf::Font font;
00017     sf::Text menu[5];
00019     sf::Texture texture;
00021     sf::Sprite background;
00022
00023 public:
00029     Menu(int k_time, std::string f_path, std::filesystem::path backgrd_path);
00031     ~Menu();
00036     void draw(sf::RenderWindow& window);
00038     virtual void setValues();
00042     void moveUp(int min_opt);
00044     virtual void moveDown();
00046     int &getKeytime();
00048     int getSelectedItem();
00050     sf::Font &getFont();
00052     void setText(int i, sf::Text text);
00054     sf::Sprite &getBackground();
00056     int &getSelectedItemIndex();
00058     sf::Text& getMenuText(int i);
00059 };
```

5.8 Player.h

```
00001 #pragma once
00002 #include "Bullet.h"
00003 #include "Entity.h"
00004 #include "HUD.h"
00005 #include "SFML\Audio.hpp"
00006 #include <future>
00007
00011 class Player : public Entity
00012 {
00013 private:
00015     int max_ammo;
00017     int curr_ammo;
00019     int temp_ammo_for_hud;
00021     std::vector<sf::RectangleShape> bullet_clip;
00023     int shootAnimationTimer = 0;
00025     bool isShooting = false;
00027     bool isReloading = false;
00029     sf::Vector2f start_pos;
00031     int sht_timer;
00032
00033 public:
00035     Player();
00042     Player(int lives_ct, int max_amm, int curr_amm, sf::Vector2f pos);
00044     ~Player();
```

```
00046     void resetAmmo(HUD& hud);
00050     void player1Movement(int windsize);
00054     void player2Movement(int windsize);
00059     bool playerReload(sf::Clock& reloadClock, sf::Keyboard::Key key);
00068     bool shootPistol(sf::Keyboard::Key shootkey, Bullet& bullet, sf::RenderWindow& target_window,
      sf::Clock& reloadClock, HUD& hud);
00070     int getBulletClipSize();
00072     std::vector<sf::RectangleShape>& getBulletClip();
00074     int getCurrAmmo();
00076     int getMaxAmmo();
00078     bool getIsShooting();
00080     bool getIsReloading();
00084     void changeIsShootingValue(bool val);
00086     void playerReset();
00090     void shootingAnimation(sf::Clock reloadclock);
00095     void reloadAnimation(sf::Clock reloadclock, HUD &hud);
00099     void Draw(sf::RenderWindow& target);
00101     int getShotTimer();
00102 };
```

5.9 resource.h

```
00001 //{{NO_DEPENDENCIES}}
00002 // Microsoft Visual C++ generated include file.
00003 // Used by Resource.rc
00004
00005 // Naste wartoci domylne dla nowych obiekt
00006 //
00007 #ifdef APSTUDIO_INVOKED
00008 #ifndef APSTUDIO_READONLY_SYMBOLS
00009 #define _APS_NEXT_RESOURCE_VALUE        101
00010 #define _APS_NEXT_COMMAND_VALUE         40001
00011 #define _APS_NEXT_CONTROL_VALUE         1001
00012 #define _APS_NEXT_SYMED_VALUE           101
00013 #endif
00014 #endif
```

5.10 resource1.h

```
00001 //{{NO_DEPENDENCIES}}
00002 // Plik doczany wygenerowany przez rodowisko Microsoft Visual C++.
00003 // Uywany przez: Resource.rc
00004 //
00005 #define IDI_ICON1                       101
00006
00007 // Next default values for new objects
00008 //
00009 #ifdef APSTUDIO_INVOKED
00010 #ifndef APSTUDIO_READONLY_SYMBOLS
00011 #define _APS_NEXT_RESOURCE_VALUE        102
00012 #define _APS_NEXT_COMMAND_VALUE         40001
00013 #define _APS_NEXT_CONTROL_VALUE         1001
00014 #define _APS_NEXT_SYMED_VALUE           101
00015 #endif
00016 #endif
```

5.11 Zombie.h

```
00001 #pragma once
00002 #include "Entity.h"
00003 #include "Player.h"
00004
00008 class Zombie : public Entity
00009 {
00010 private:
00012     sf::Clock spawn_clock;
00014     int score_value;
00016     int base_spawn_freq;
00018     int curr_spawn_freq;
00020     int basic_zombie_count = 0;
00022     int curr_lives = Zombie::Entity::getLives();
00024     float base_mov_speed;
00026     float curr_mov_speed;
00027 public:
00029     Zombie();
```

```
00037     Zombie(int score_val, int lives_ct, int spawn_freq_ms, sf::Vector2f pos, float mov_spd);
00039     ~Zombie();
00041     float getMovSpeed();
00043     int getCurrSpawnFreq();
00045     void setCurrSpawnFreq(int val);
00047     int getBaseSpawnFreq();
00049     void setBaseSpawnFreq(int val);
00051     void setCurrMovSpeed(int val);
00053     void resetValues();
00055     void restartClock();
00057     sf::Clock &getSpawnClock();
00059     int getScoreValue();
00060 };
```

Index