

## My Project

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>README</b>	<b>1</b>
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>7</b>
4.1	Boss Class Reference . . . . .	7
4.2	Character Class Reference . . . . .	7
4.2.1	Constructor & Destructor Documentation . . . . .	8
4.2.1.1	Character() . . . . .	8
4.2.1.2	~Character() . . . . .	8
4.2.2	Member Function Documentation . . . . .	8
4.2.2.1	gravity() . . . . .	8
4.2.2.2	take_damage() . . . . .	8
4.2.3	Member Data Documentation . . . . .	9
4.2.3.1	damage_modifier . . . . .	9
4.2.3.2	fall_speed . . . . .	9
4.2.3.3	falling . . . . .	9
4.2.3.4	health_points . . . . .	9
4.2.3.5	on_platform . . . . .	9
4.2.3.6	shoot_clock . . . . .	9
4.3	GameEngine Struct Reference . . . . .	10

4.3.1	Constructor & Destructor Documentation . . . . .	10
4.3.1.1	GameEngine() . . . . .	10
4.3.2	Member Function Documentation . . . . .	10
4.3.2.1	run() . . . . .	10
4.4	Object Class Reference . . . . .	10
4.4.1	Constructor & Destructor Documentation . . . . .	11
4.4.1.1	Object() . . . . .	11
4.4.2	Member Data Documentation . . . . .	11
4.4.2.1	texture . . . . .	11
4.5	Platform Class Reference . . . . .	11
4.5.1	Constructor & Destructor Documentation . . . . .	12
4.5.1.1	Platform() . . . . .	12
4.5.2	Member Function Documentation . . . . .	12
4.5.2.1	render() . . . . .	12
4.5.2.2	update() . . . . .	12
4.6	Player Class Reference . . . . .	13
4.6.1	Constructor & Destructor Documentation . . . . .	13
4.6.1.1	Player() . . . . .	13
4.6.2	Member Function Documentation . . . . .	13
4.6.2.1	check_parry() . . . . .	13
4.6.2.2	render() . . . . .	13
4.6.2.3	shoot() . . . . .	14
4.6.2.4	update() . . . . .	14
4.7	PlayState Class Reference . . . . .	14
4.7.1	Constructor & Destructor Documentation . . . . .	14
4.7.1.1	PlayState() . . . . .	15
4.7.1.2	~PlayState() . . . . .	15
4.7.2	Member Function Documentation . . . . .	15
4.7.2.1	display_loss() . . . . .	15
4.7.2.2	display_win() . . . . .	15

---

4.7.2.3	draw_screen()	15
4.7.2.4	update_gravity()	15
4.7.2.5	update_screen()	16
4.8	Projectile Class Reference	16
4.8.1	Constructor & Destructor Documentation	16
4.8.1.1	Projectile()	16
4.8.2	Member Function Documentation	17
4.8.2.1	invert_direction()	17
4.8.2.2	render()	17
4.8.2.3	update()	17
4.8.3	Member Data Documentation	17
4.8.3.1	damage_modifier	17
4.8.3.2	direction	17
4.8.3.3	height	18
4.8.3.4	width	18
<b>Index</b>		<b>19</b>



## **Chapter 1**

# **README**





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

GameEngine . . . . .	10
PlayState . . . . .	14
Sprite	
Object . . . . .	10
Character . . . . .	7
Boss . . . . .	7
Player . . . . .	13
Platform . . . . .	11
Projectile . . . . .	16



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Boss</a>	7
<a href="#">Character</a>	7
<a href="#">GameEngine</a>	10
<a href="#">Object</a>	10
<a href="#">Platform</a>	11
<a href="#">Player</a>	13
<a href="#">PlayState</a>	14
<a href="#">Projectile</a>	16



## Chapter 4

# Class Documentation

### 4.1 Boss Class Reference

Inheritance diagram for Boss:

### 4.2 Character Class Reference

Inheritance diagram for Character:

Collaboration diagram for Character:

#### Public Member Functions

- [Character](#) (double, int, std::string)
- [~Character](#) ()
- bool **is\_alive** () const
- virtual void **resurrect** ()
- virtual void **shoot** (std::vector< [Object](#) \*> &)=0
- void [take\\_damage](#) (double const)
- void [gravity](#) (float const)
- void **set\_falling** (bool const &)
- void **set\_fall\_speed** (int)
- int **get\_fall\_speed** () const
- void **set\_on\_platform** (bool const)

#### Protected Member Functions

- double **get\_health\_points** () const
- int **get\_damage\_modifier** () const

## Protected Attributes

- bool [falling](#) {}
- bool [on\\_platform](#) {}
- double [health\\_points](#) {}
- int [damage\\_modifier](#) {}
- int [fall\\_speed](#) {}
- sf::Clock [shoot\\_clock](#) {}

## 4.2.1 Constructor & Destructor Documentation

### 4.2.1.1 [Character\(\)](#)

```
Character::Character (
    double health_points = 0,
    int damage_modifier = 0,
    std::string texture_filename = "static/default.png" )
```

[Character](#) constructor implementation.

### 4.2.1.2 [~Character\(\)](#)

```
Character::~~Character ( )
```

[Character](#) destructor implementation.

## 4.2.2 Member Function Documentation

### 4.2.2.1 [gravity\(\)](#)

```
void Character::gravity (
    float const height )
```

Checks if character is not standing on the ground or a platform. If it's not standing on the ground or platform we want to move it down to simulate gravity.

### 4.2.2.2 [take\\_damage\(\)](#)

```
void Character::take_damage (
    double const damage )
```

Used to remove `health_points` from a character.

### 4.2.3 Member Data Documentation

#### 4.2.3.1 damage\_modifier

```
int Character::damage_modifier {} [protected]
```

Stores damage output of projectiles.

#### 4.2.3.2 fall\_speed

```
int Character::fall_speed {} [protected]
```

Stores how fast character is falling.

#### 4.2.3.3 falling

```
bool Character::falling {} [protected]
```

Stores if character is falling or not.

#### 4.2.3.4 health\_points

```
double Character::health_points {} [protected]
```

Stores health\_points.

#### 4.2.3.5 on\_platform

```
bool Character::on_platform {} [protected]
```

Stores if character is on platform.

#### 4.2.3.6 shoot\_clock

```
sf::Clock Character::shoot_clock {} [protected]
```

Clock for buffering shots.

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

- Character.h
- Character.cc

## 4.3 GameEngine Struct Reference

### Public Member Functions

- [GameEngine](#) ()
- int [run](#) ()

#### 4.3.1 Constructor & Destructor Documentation

##### 4.3.1.1 GameEngine()

```
GameEngine::GameEngine ( )
```

[GameEngine](#) constructor implementation.

#### 4.3.2 Member Function Documentation

##### 4.3.2.1 run()

```
int GameEngine::run ( )
```

Implementation of GameEngines run function. We create a new PlayState\* called current\_game. We then use check\_win and check\_loss to check if the game is won or lost every frame. If it is either we call display\_win or display\_loss. If the game is not won or lost we call for update\_screen and update\_gravity and lastly we draw the screen with draw\_screen.

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

- GameEngine.h
- GameEngine.cc

## 4.4 Object Class Reference

Inheritance diagram for Object:

Collaboration diagram for Object:

### Public Member Functions

- [Object](#) (std::string)
- virtual void **render** (sf::RenderWindow &)=0
- virtual void **update** (std::vector< [Object](#) \*> &, [Object](#) \*, [Object](#) \*)=0



## Protected Attributes

- sf::Texture [texture](#) {}

## 4.4.1 Constructor & Destructor Documentation

### 4.4.1.1 Object()

```
Object::Object (
    std::string sprite_name )
```

[Object](#) constructor implementation.

## 4.4.2 Member Data Documentation

### 4.4.2.1 texture

```
sf::Texture Object::texture {} [protected]
```

Stores the texture for the object.

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

- Object.h
- Object.cc

## 4.5 Platform Class Reference

Inheritance diagram for Platform:

Collaboration diagram for Platform:

## Public Member Functions

- [Platform](#) (float, float, double, double)
- void [render](#) (sf::RenderWindow &) override
- void [update](#) (std::vector< [Object](#) \*> &, [Object](#) \*, [Object](#) \*) override

## Additional Inherited Members

### 4.5.1 Constructor & Destructor Documentation

#### 4.5.1.1 Platform()

```
Platform::Platform (
    float x = 0,
    float y = 0,
    double width = 0,
    double height = 0 )
```

[Platform](#) constructor implementation.

### 4.5.2 Member Function Documentation

#### 4.5.2.1 render()

```
void Platform::render (
    sf::RenderWindow & drawto ) [override], [virtual]
```

Draws the platform on the screen.

Implements [Object](#).

#### 4.5.2.2 update()

```
void Platform::update (
    std::vector< Object *> & ,
    Object * ,
    Object * ) [override], [virtual]
```

update is run every frame and checks whether or not there is a player or boss standing on it.

Implements [Object](#).

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

- Platform.h
- Platform.cc

## 4.6 Player Class Reference

Inheritance diagram for Player:

Collaboration diagram for Player:

### Public Member Functions

- [Player](#) ()
- void [shoot](#) (std::vector< [Object](#) \*> &) override
- void [update](#) (std::vector< [Object](#) \*> &, [Object](#) \*, [Object](#) \*) override
- void [render](#) (sf::RenderWindow &) override
- bool [check\\_parry](#) ([Object](#) \*, [Projectile](#) \*) const

### Additional Inherited Members

#### 4.6.1 Constructor & Destructor Documentation

##### 4.6.1.1 [Player](#)()

```
Player::Player ( )
```

[Player](#) constructor implementation.

#### 4.6.2 Member Function Documentation

##### 4.6.2.1 [check\\_parry](#)()

```
bool Player::check_parry (
    Object * player,
    Projectile * proj ) const
```

Checks if a projectile is within a certain distance (200.f here) in front of the player. If it is inside this area and the parry timer for player is higher than 3 the function returns true.

##### 4.6.2.2 [render](#)()

```
void Player::render (
    sf::RenderWindow & drawto ) [override], [virtual]
```

Draws the player and player-hp on the screen.

Implements [Object](#).

#### 4.6.2.3 shoot()

```
void Player::shoot (
    std::vector< Object *> & objects ) [override], [virtual]
```

Creates a new object of type [Projectile](#) in front of players position. We then insert this object into a vector of objects.

Implements [Character](#).

#### 4.6.2.4 update()

```
void Player::update (
    std::vector< Object *> & objects,
    Object * player,
    Object * boss ) [override], [virtual]
```

update is run every frame and checks for keypresses that control moving, shooting and parrying. It also checks for collision between player and boss.

Implements [Object](#).

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

- [Player.h](#)
- [Player.cc](#)

## 4.7 PlayState Class Reference

### Public Member Functions

- [PlayState](#) (int)
- [~PlayState](#) ()
- void [update\\_screen](#) ()
- void [update\\_gravity](#) (float const)
- void [draw\\_screen](#) (sf::RenderWindow &)
- bool [check\\_win](#) () const
- bool [check\\_loss](#) () const
- void [display\\_win](#) (sf::RenderWindow &) const
- void [display\\_loss](#) (sf::RenderWindow &) const

#### 4.7.1 Constructor & Destructor Documentation

#### 4.7.1.1 PlayState()

```
PlayState::PlayState (
    int stage = 1 )
```

[PlayState](#) constructor implementation.

#### 4.7.1.2 ~PlayState()

```
PlayState::~~PlayState ( )
```

Playstate destructor implementation.

### 4.7.2 Member Function Documentation

#### 4.7.2.1 display\_loss()

```
void PlayState::display_loss (
    sf::RenderWindow & drawto ) const
```

[PlayState](#) implementation of display\_loss. Draws the lose screen if the player lost.

#### 4.7.2.2 display\_win()

```
void PlayState::display_win (
    sf::RenderWindow & drawto ) const
```

[PlayState](#) implementation of display\_win. Draws the win screen if the player won.

#### 4.7.2.3 draw\_screen()

```
void PlayState::draw_screen (
    sf::RenderWindow & drawto )
```

[PlayState](#) implementation of draw\_screen. Draws everything that is shown on the screen.

#### 4.7.2.4 update\_gravity()

```
void PlayState::update_gravity (
    float const height )
```

[PlayState](#) implementation of update\_gravity. Calls the gravity function for player and boss.

#### 4.7.2.5 update\_screen()

```
void PlayState::update_screen ( )
```

[PlayState](#) implementation of update\_screen. Calls the update function on all onscreen Objects.

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

- PlayState.h
- PlayState.cc

## 4.8 Projectile Class Reference

Inheritance diagram for Projectile:

Collaboration diagram for Projectile:

### Public Member Functions

- [Projectile](#) (int const, float const, float const, float const, float const, double const)
- void [render](#) (sf::RenderWindow &) override
- void [update](#) (std::vector< [Object](#) \*> &, [Object](#) \*, [Object](#) \*) override
- float [get\\_height](#) () const
- float [get\\_width](#) () const
- int [invert\\_direction](#) ()
- int [get\\_direction](#) () const

### Protected Attributes

- int [direction](#) {}
- double const [damage\\_modifier](#) {}
- float const [height](#) {}
- float const [width](#) {}

### 4.8.1 Constructor & Destructor Documentation

#### 4.8.1.1 Projectile()

```
Projectile::Projectile (
    int const direction,
    float const width,
    float const height,
    float const x,
    float const y,
    double const damage_modifier )
```

[Projectile](#) constructor implementation.

## 4.8.2 Member Function Documentation

### 4.8.2.1 invert\_direction()

```
int Projectile::invert_direction ( )
```

Inverts the direction the projectile is going in, and flips the sprite.

### 4.8.2.2 render()

```
void Projectile::render (
    sf::RenderWindow & drawto ) [override], [virtual]
```

Draws the projectile on the screen.

Implements [Object](#).

### 4.8.2.3 update()

```
void Projectile::update (
    std::vector< Object *> & objects,
    Object * player,
    Object * boss ) [override], [virtual]
```

[Projectile](#)'s implementation of update. Despawns this if it hits the player or boss, or goes out of bounds.

Implements [Object](#).

## 4.8.3 Member Data Documentation

### 4.8.3.1 damage\_modifier

```
double const Projectile::damage_modifier {} [protected]
```

Stores how much damage the projectile does.

### 4.8.3.2 direction

```
int Projectile::direction {} [protected]
```

Stores if the projectile goes left or right.

#### 4.8.3.3 height

```
float const Projectile::height {} [protected]
```

Stores the height of the projectile.

#### 4.8.3.4 width

```
float const Projectile::width {} [protected]
```

Stores the width of the projectile.

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

- Projectile.h
- Projectile.cc



# Index

~Character  
    Character, 8  
~PlayState  
    PlayState, 15

Boss, 7

Character, 7  
    ~Character, 8  
    Character, 8  
    damage\_modifier, 9  
    fall\_speed, 9  
    falling, 9  
    gravity, 8  
    health\_points, 9  
    on\_platform, 9  
    shoot\_clock, 9  
    take\_damage, 8

check\_parry  
    Player, 13

damage\_modifier  
    Character, 9  
    Projectile, 17

direction  
    Projectile, 17

display\_loss  
    PlayState, 15

display\_win  
    PlayState, 15

draw\_screen  
    PlayState, 15

fall\_speed  
    Character, 9

falling  
    Character, 9

GameEngine, 10  
    GameEngine, 10  
    run, 10

gravity  
    Character, 8

health\_points  
    Character, 9

height  
    Projectile, 17

invert\_direction  
    Projectile, 17

Object, 10  
    Object, 11  
    texture, 11  
on\_platform  
    Character, 9

Platform, 11  
    Platform, 12  
    render, 12  
    update, 12  
PlayState, 14  
    ~PlayState, 15  
    display\_loss, 15  
    display\_win, 15  
    draw\_screen, 15  
    PlayState, 14  
    update\_gravity, 15  
    update\_screen, 15

Player, 13  
    check\_parry, 13  
    Player, 13  
    render, 13  
    shoot, 13  
    update, 14

Projectile, 16  
    damage\_modifier, 17  
    direction, 17  
    height, 17  
    invert\_direction, 17  
    Projectile, 16  
    render, 17  
    update, 17  
    width, 18

render  
    Platform, 12  
    Player, 13  
    Projectile, 17

run  
    GameEngine, 10

shoot  
    Player, 13

shoot\_clock  
    Character, 9

take\_damage  
    Character, 8

texture  
    Object, 11

- update
  - Platform, [12](#)
  - Player, [14](#)
  - Projectile, [17](#)
- update\_gravity
  - PlayState, [15](#)
- update\_screen
  - PlayState, [15](#)
- width
  - Projectile, [18](#)