My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

README

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

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 fron 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