Dungeon Crawler Group 15

Generated by Doxygen 1.8.17

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

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

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 Animation Class Reference

`#include <animation.hpp>`

Public Member Functions

- Animation (const std::string sprite_sheets_path, int frame_x_pos, int frame_y_pos)
- ~Animation ()
- void setAnimationSpeed (float new_speed)
- sf::Texture * getCurrentFrame ()
- void resetCurrentFrame ()
- bool Update (float time)

Public Attributes

- std::vector< sf::Texture > animation_frames_
- unsigned int current_frame_ = 0
- unsigned int last_frame_
- float animation_speed_ = 0.2
- float actionTimer_

5.1.1 Detailed Description

Definition at line 10 of file animation.hpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Animation()

```
Animation::Animation (
            const std::string sprite_sheets_path,
            int frame_x_pos,
            int frame_y_pos )
```

Animation constructor

Definition at line 18 of file animation.cpp.

```
00018
            {
00019   sf::Texture temp_texture;
00020   temp_texture.loadFromFile(sprite_sheets_path);
00021   sf::Vector2u texture_size = temp_texture.getSize();
00022   int amount_of_x_offsets = (int(texture_size.x) / texture_crop_scale_x_);
00023   int amount_of_y_offsets = (int(texture_size.y) / texture_crop_scale_y_);
00024   int x_offset = 0; int y_offset = 0;
00025   for (int j = 0; j < amount_of_y_offsets; j++) {
00026     y_offset = j*texture_crop_scale_y_;
00027     for (int i = 0; i < amount_of_x_offsets; i++) {
00028       x_offset = i*texture_crop_scale_x_;
00029       sf::IntRect cropping(x_offset, y_offset, texture_crop_scale_x_, texture_crop_scale_y_);
00030       sf::Texture frame_texture;
00031       frame_texture.loadFromFile(sprite_sheets_path, cropping);
00032       animation_frames_.push_back(frame_texture);
00033     }
00034   }
00035   last_frame_ = animation_frames_.size() - 1;
00036 };
```

5.1.2.2 ~Animation()

```
Animation::~Animation ( )
```

Default destructor

Definition at line 39 of file animation.cpp.

```
00039 {};
```

5.1.3 Member Function Documentation

5.1.3.1 getCurrentFrame()

```
sf::Texture * Animation::getCurrentFrame ( )
```

Returns current frame of the animation image as texture

Definition at line 49 of file animation.cpp.

```
00049 {return &animation_frames_[current_frame_];};
```

5.1.3.2 resetCurrentFrame()

```
void Animation::resetCurrentFrame ( )
```

Resets the current frame

Definition at line 55 of file animation.cpp.

```
00055 {current_frame_ = 0;};
```

5.1.3.3 setAnimationSpeed()

```
void Animation::setAnimationSpeed (
            float new_speed )
```

Sets the animation speed to new_speed value

Definition at line 42 of file animation.cpp.

```
00042 {animation_speed_ = new_speed;};
```

5.1.3.4 Update()

```
bool Animation::Update (
            float time )
```

Updates the frame

Definition at line 62 of file animation.cpp.

```
00062                                    {
00063   actionTimer_ += time;
00064   if(actionTimer_ >= animation_speed_){
00065     actionTimer_ = 0;
00066     current_frame_ += 1;
00067     if(current_frame_ > last_frame_){
00068       current_frame_ = 0;
00069     }
00070     return true;
00071   }
00072   return false;
00073 };
```

5.1.4 Member Data Documentation

5.1.4.1 actionTimer_

```
float Animation::actionTimer_
```

Definition at line 29 of file animation.hpp.

5.1.4.2 animation_frames_

```
std::vector<sf::Texture> Animation::animation_frames_
```

Definition at line 25 of file animation.hpp.

5.1.4.3 animation_speed_

```
float Animation::animation_speed_ = 0.2
```

Definition at line 28 of file animation.hpp.

5.1.4.4 current_frame_

```
unsigned int Animation::current_frame_ = 0
```

Definition at line 26 of file animation.hpp.

5.1.4.5 last_frame_

```
unsigned int Animation::last_frame_
```

Definition at line 27 of file animation.hpp.

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

- src/animation.hpp
- src/animation.cpp

5.2 ChasingEnemy Class Reference

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

Inheritance diagram for ChasingEnemy:

Collaboration diagram for ChasingEnemy:

Public Member Functions

- ChasingEnemy (Animation *enemy_animation, sf::Vector2f spawnPos, float speed)
- ~ChasingEnemy ()
- virtual void Update (float time, sf::Vector2f player_position) override

Additional Inherited Members

5.2.1 Detailed Description

Class for enemy that chases the player

Definition at line 61 of file enemy.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ChasingEnemy()

```
ChasingEnemy::ChasingEnemy (
            Animation * enemy_animation,
            sf::Vector2f spawnPos,
            float speed )
```

Definition at line 54 of file enemy.cpp.

```
00055   : Enemy(enemy_animation, spawnPos, speed) {};
```

5.2.2.2 ~ChasingEnemy()

```
ChasingEnemy::~ChasingEnemy ( )
```

Definition at line 57 of file enemy.cpp.

```
00057 {};
```

5.2.3 Member Function Documentation

5.2.3.1 Update()

```
void ChasingEnemy::Update (
            float time,
            sf::Vector2f player_position )  [override], [virtual]
```

Virtual function for updating enemy position

Implements Enemy.

Definition at line 59 of file enemy.cpp.

```
00059                                                                {
00060   currentSpeed_ = speed_;
00061   actionTimer_ += time;
00062   //std::cout « "Interval timer: " « actionTimer_ « "\n";
00063
00064   /* Change walk direction to some random direction at random intervals */
00065
00066   if(actionTimer_ >= walkInterval_){
00067     actionTimer_ = 0.0f;
00068     walk_ = walk_ ? false : true; //tenary operator
00069
00070     /*
00071     walkInterval_ is a float between 0.5 and 2.5,
00072     this is how it is generated:
00073
00074     rand() gives a number between 0 and RAND_MAX,
00075     so divide by RAND_MAX to get a float between 0 and 1,
00076     or in this case 0 and 2, because it is multiplied by 2,
00077     and then plus 0.5 of that to get float between 0.5 and 2.5.
00078     */
00079     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00080
00081     if(walk_ == false){
00082       enemy_animations_[0].resetCurrentFrame();
00083       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00084     } else{
00085       attacking_player_ = (rand() % 3) == 1; //Enemies randomly attack sometimes
00086       walkInterval_ = 0.7;
00087     }
00088
00089     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00090     if(attacking_player_ == true) {
00091       sf::Vector2f direction_vector = player_position - body_.getPosition();
00092       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00093       direction_vector.x = direction_vector.x / vector_magnitude;
00094       direction_vector.y = direction_vector.y / vector_magnitude;
00095       //std::cout « direction_vector.x « " " « direction_vector.y « " " « vector_magnitude « "\n";
00096       movement.x *= time * direction_vector.x * 2.f;
00097       movement.y *= time * direction_vector.y * 2.f;
00098     } else {
00099       float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00100       float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00101       movement.x *= random_float_x * time;
00102       movement.y *= random_float_y * time;
00103     }
00104     direction_ = movement;
00105   }
00106   if(walk_){
00107     if(enemy_animations_[0].Update(time)){
00108       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00109     }
00110     body_.move(direction_);
00111     /*Collider col = this->getCollider();
00112     for (auto room : rooms) {
00113       for (auto wall : room.walls) {
00114         for (auto w : wall) {
00115           if (w.type_ == 1) {
00116             Collider wallCol = w.getCollider();
00117             wallCol.checkCollider(col, 1);
00118           }
00119         }
00120       }
00121     }*/
00122   }
00123 };
```

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

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

5.3 Collider Class Reference

`#include <collider.hpp>`

Public Member Functions

- Collider (sf::RectangleShape &body)
- ~Collider ()
- void Move (float dx, float dy)
- bool checkCollider (Collider &other, float push)
- sf::Vector2f getPosition ()
- sf::Vector2f getHalfSize ()

5.3.1 Detailed Description

Class for handling collisions

Definition at line 6 of file collider.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Collider()

```
Collider::Collider (
            sf::RectangleShape & body )
```

Constructor

Definition at line 3 of file collider.cpp.

```
00003 : body_(body) {}
```

5.3.2.2 ~Collider()

```
Collider::~Collider ( )
```

Default destructor

Definition at line 5 of file collider.cpp.

```
00005 {}
```

5.3.3 Member Function Documentation

5.3.3.1 checkCollider()

```
bool Collider::checkCollider (
            Collider & other,
            float push )
```

Checks if two colliders are colliding

Definition at line 9 of file collider.cpp.

```
00009                                                       {
00010   // get positions
00011   sf::Vector2f otherPosition = other.getPosition();
00012   sf::Vector2f otherHalfSize = other.getHalfSize();
00013   sf::Vector2f thisPosition = getPosition();
00014   sf::Vector2f thisHalfSize = getHalfSize();
00015
00016   // find differences in location
00017   float xdif = otherPosition.x - thisPosition.x;
00018   float ydif = otherPosition.y - thisPosition.y;
00019
00020   float intersectX = abs(xdif) - (otherHalfSize.x + thisHalfSize.x);
00021   float intersectY = abs(ydif) - (otherHalfSize.y + thisHalfSize.y);
00022
00023   // if colliders intersect, move one of them based on push parameter and return
00024   // true
00025   if (intersectX < 0.0f && intersectY < 0.0f) {
00026     push = std::min(std::max(push, 0.0f), 1.0f);
00027
00028     if (abs(intersectX) < abs(intersectY)) {
00029       if (xdif > 0.0f) {
00030         Move(intersectX * (1.0f - push), 0.0f);
00031         other.Move(-intersectX * push, 0.0f);
00032       } else {
00033         Move(-intersectX * (1.0f - push), 0.0f);
00034         other.Move(intersectX * push, 0.0f);
00035       }
00036     } else {
00037       if (ydif > 0.0f) {
00038         Move(0.0f, intersectY * (1.0f - push));
00039         other.Move(0.0f, -intersectY * push);
00040       } else {
00041         Move(0.0f, -intersectY * (1.0f - push));
00042         other.Move(0.0f, intersectY * push);
00043       }
00044     }
00045
00046     return true;
00047   }
00048
00049   return false;
00050 }
```

5.3.3.2 getHalfSize()

```
sf::Vector2f Collider::getHalfSize ( )  [inline]
```

Returns collider size divided by 2

Definition at line 19 of file collider.hpp.

```
00019 { return body_.getSize() / 2.0f; }
```

5.3.3.3 getPosition()

```
sf::Vector2f Collider::getPosition ( )  [inline]
```

Returns collider position

Definition at line 17 of file collider.hpp.

```
00017 { return body_.getPosition(); }
```

5.3.3.4 Move()

```
void Collider::Move (
             float dx,
             float dy )  [inline]
```

Moves object out of collision

Definition at line 13 of file collider.hpp.
```
00013 { body_.move(dx, dy); }
```

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

- src/collider.hpp
- src/collider.cpp

5.4 Enemy Class Reference

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

Inheritance diagram for Enemy:

Public Member Functions

- Enemy (Animation *enemy_animation, sf::Vector2f spawnPos, float speed)
- ~Enemy ()
- sf::Vector2f getPosition ()
- bool isRanged ()
- Collider getCollider ()
- void MoveRoom (int dir)
- void Draw (sf::RenderWindow &window)
- void Damage (float dmg)
- virtual void Update (float time, sf::Vector2f player_position)=0
- virtual void Update2 (float time, sf::Vector2f player_position, std::vector< Projectile > &active_projectiles)

Public Attributes

- bool walk_ = false
- bool attacking_player_ = false
- bool is_ranged_type_ = false
- float speed_ = 100
- float hp_ = 100
- float damage_ = 1
- float currentSpeed_
- float actionTimer_
- float walkInterval_ = 2
- bool alive_ = true
- sf::RectangleShape body_
- sf::Vector2f direction_
- std::vector< Animation > enemy_animations_

5.4.1 Detailed Description

Virtual class for enemies

Definition at line 17 of file enemy.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Enemy()

```
Enemy::Enemy (
            Animation * enemy_animation,
            sf::Vector2f spawnPos,
            float speed )
```

Constructor

Definition at line 10 of file enemy.cpp.

```
00011     : speed_(speed) {
00012   body_.setSize(sf::Vector2f(40.0f, 40.0f));
00013   body_.setOrigin(body_.getSize() / 2.0f);
00014   body_.setPosition(spawnPos);
00015   enemy_animations_.push_back(*enemy_animation);
00016   body_.setTexture(&(enemy_animation->animation_frames_[0]));
00017 };
```

5.4.2.2 ~Enemy()

```
Enemy::~Enemy ( )
```

Default destructor

Definition at line 19 of file enemy.cpp.

```
00019 {};
```

5.4.3 Member Function Documentation

5.4.3.1 Damage()

```
void Enemy::Damage (
            float dmg )
```

Reduces enemy hp

Definition at line 27 of file enemy.cpp.

```
00027                                {
00028   hp_ -= damage;
00029   if (hp_ <= 0) {alive_ = false;} //std::cout « "The NPC has died!" « std::endl;
00030 }
```

5.4.3.2 Draw()

```
void Enemy::Draw (
            sf::RenderWindow & window )
```

Used for rendering the enemy

Definition at line 25 of file enemy.cpp.

```
00025 { window.draw(body_); };
```

5.4.3.3 getCollider()

```
Collider Enemy::getCollider ( )  [inline]
```

Returns enemy collider

Definition at line 28 of file enemy.hpp.

```
00029   {
00030     return Collider(body_);
00031   };
```

5.4.3.4 getPosition()

```
sf::Vector2f Enemy::getPosition ( )
```

Returns enemy position

Definition at line 21 of file enemy.cpp.

```
00021 { return body_.getPosition(); };
```

5.4.3.5 isRanged()

```
bool Enemy::isRanged ( )
```

Check if enemy is ranged

Definition at line 23 of file enemy.cpp.

```
00023 { return is_ranged_type_; };
```

5.4.3.6 MoveRoom()

```
void Enemy::MoveRoom (
            int dir )
```

Moves enemy when player is traversing between rooms

Definition at line 34 of file enemy.cpp.

```
00034                                {
00035   sf::Vector2f dir;
00036   if (direction == 0) {
00037     dir = sf::Vector2f(0, -10 * 50);
00038   }
00039   if (direction == 1) {
00040     dir = sf::Vector2f(10 * 50, 0);
00041   }
00042   if (direction == 2) {
00043     dir = sf::Vector2f(0, 10 * 50);
00044   }
00045   if (direction == 3) {
00046     dir = sf::Vector2f(-10 * 50, 0);
00047   }
00048   body_.move(dir);
00049 }
```

5.4.3.7 Update()

```
virtual void Enemy::Update (
            float time,
            sf::Vector2f player_position )  [pure virtual]
```

Virtual function for updating enemy position

Implemented in FinalBoss, RangedEnemy, and ChasingEnemy.

5.4.3.8 Update2()

```
void Enemy::Update2 (
            float time,
            sf::Vector2f player_position,
            std::vector< Projectile > & active_projectiles )  [virtual]
```

Reimplemented in FinalBoss, and RangedEnemy.

Definition at line 32 of file enemy.cpp.

```
00032 {};
```

5.4.4 Member Data Documentation

5.4.4.1 actionTimer_

```
float Enemy::actionTimer_
```

Definition at line 50 of file enemy.hpp.

5.4.4.2 alive_

```
bool Enemy::alive_ = true
```

Definition at line 52 of file enemy.hpp.

5.4.4.3 attacking_player_

```
bool Enemy::attacking_player_ = false
```

Definition at line 44 of file enemy.hpp.

5.4.4.4 body_

```
sf::RectangleShape Enemy::body_
```

Definition at line 53 of file enemy.hpp.

5.4.4.5 currentSpeed_

```
float Enemy::currentSpeed_
```

Definition at line 49 of file enemy.hpp.

5.4.4.6 damage_

```
float Enemy::damage_ = 1
```

Definition at line 48 of file enemy.hpp.

5.4.4.7 direction_

```
sf::Vector2f Enemy::direction_
```

Definition at line 54 of file enemy.hpp.

5.4.4.8 enemy_animations_

```
std::vector<Animation> Enemy::enemy_animations_
```

Definition at line 55 of file enemy.hpp.

5.4.4.9 hp_

```
float Enemy::hp_ = 100
```

Definition at line 47 of file enemy.hpp.

5.4.4.10 is_ranged_type_

```
bool Enemy::is_ranged_type_ = false
```

Definition at line 45 of file enemy.hpp.

5.4.4.11 speed_

```
float Enemy::speed_ = 100
```

Definition at line 46 of file enemy.hpp.

5.4.4.12 walk_

```
bool Enemy::walk_ = false
```

Definition at line 43 of file enemy.hpp.

5.4.4.13 walkInterval_

```
float Enemy::walkInterval_ = 2
```

Definition at line 51 of file enemy.hpp.

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

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

5.5 FinalBoss Class Reference

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

Inheritance diagram for FinalBoss:

Collaboration diagram for FinalBoss:

Public Member Functions

- FinalBoss (Animation *enemy_animation, sf::Vector2f spawnPos, float speed, sf::Texture *enemy_↩ projectile_texture)
- virtual void Update (float time, sf::Vector2f player_position) override
- void Update2 (float time, sf::Vector2f player_position, std::vector< Projectile > &active_projectiles)
- Projectile ShootAtDirection (sf::Vector2f direction)

Public Attributes

- sf::Texture * enemy_projectile_texture_

5.5.1 Detailed Description

Class for the finall Boss

Definition at line 94 of file enemy.hpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 FinalBoss()

```
FinalBoss::FinalBoss (
            Animation * enemy_animation,
            sf::Vector2f spawnPos,
            float speed,
            sf::Texture * enemy_projectile_texture )
```

Definition at line 277 of file enemy.cpp.

```
00279     : Enemy(enemy_animation, spawnPos, speed), enemy_projectile_texture_(enemy_projectile_texture) {
00280 is_ranged_type_ = true;
00281 walk_ = true;
00282 body_.setSize(sf::Vector2f(100.0f, 100.0f));
00283 body_.setOrigin(body_.getSize() / 2.0f);
00284 };
```

5.5.3 Member Function Documentation

5.5.3.1 ShootAtDirection()

```
Projectile FinalBoss::ShootAtDirection (
            sf::Vector2f direction )
```

Used to shoot projetiles towards direction

Definition at line 406 of file enemy.cpp.

```
00406                                                       {
00407   //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00408   //Projectile Weapon::Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory)
00409   Projectile proj(this->enemy_projectile_texture_);
00410   //proj.friendly_[0] = 0;
00411   proj.active_ = true;
00412   proj.body_.setTexture(proj.projectile_texture_);
00413   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00414   proj.body_.setPosition(body_.getPosition());
00415   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00416   proj.projectile_trajectory_ = direction/* * proj.velocity_multiplier_*/;
00417   proj.body_.move(proj.projectile_trajectory_);
00418   return proj;
00419 };
```

5.5.3.2 Update()

```
void FinalBoss::Update (
            float time,
            sf::Vector2f player_position )  [override], [virtual]
```

Virtual function for updating enemy position

Implements Enemy.

Definition at line 286 of file enemy.cpp.

```
00286                                                {
00287   currentSpeed_ = speed_;
00288   actionTimer_ += time;
00289   //std::cout << "Interval timer: " << actionTimer_ << "\n";
00290
```

```
00291   /* Change walk direction to some random direction at random intervals */
00292
00293   if(actionTimer_ >= walkInterval_){
00294     actionTimer_ = 0.0f;
00295
00296     /*
00297     walkInterval_ is a float between 0.5 and 2.5,
00298     this is how it is generated:
00299
00300     rand() gives a number between 0 and RAND_MAX,
00301     so divide by RAND_MAX to get a float between 0 and 1,
00302     or in this case 0 and 2, because it is multiplied by 2,
00303     and then plus 0.5 of that to get float between 0.5 and 2.5.
00304     */
00305     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00306
00307     attacking_player_ = (rand() % 3) == 1; //Enemies randomly attack sometimes
00308     walkInterval_ = 0.7;
00309
00310     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00311     if(attacking_player_ == true) {
00312       sf::Vector2f direction_vector = player_position - body_.getPosition();
00313       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00314       direction_vector.x = direction_vector.x / vector_magnitude;
00315       direction_vector.y = direction_vector.y / vector_magnitude;
00316       //std::cout « direction_vector.x « " " « direction_vector.y « " " « vector_magnitude « "\n";
00317       movement.x *= time * direction_vector.x * 2.f;
00318       movement.y *= time * direction_vector.y * 2.f;
00319     } else {
00320       float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00321       float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00322       movement.x *= random_float_x * time;
00323       movement.y *= random_float_y * time;
00324     }
00325     direction_ = movement;
00326   }
00327   if(enemy_animations_[0].Update(time)){
00328     body_.setTexture(enemy_animations_[0].getCurrentFrame());
00329   }
00330   body_.move(direction_);
00331     /*Collider col = this->getCollider();
00332     for (auto room : rooms) {
00333       for (auto wall : room.walls) {
00334         for (auto w : wall) {
00335           if (w.type_ == 1) {
00336             Collider wallCol = w.getCollider();
00337             wallCol.checkCollider(col, 1);
00338           }
00339         }
00340       }
00341     }*/
00342 };
```

5.5.3.3 Update2()

```
void FinalBoss::Update2 (
            float time,
            sf::Vector2f player_position,
            std::vector< Projectile > & active_projectiles )  [virtual]
```

Reimplemented from Enemy.

Definition at line 344 of file enemy.cpp.

```
00344
        {
00345   currentSpeed_ = speed_;
00346   actionTimer_ += time;
00347
00348   /* Change walk direction to some random direction at random intervals */
00349
00350   if(actionTimer_ >= walkInterval_) {
00351     actionTimer_ = 0.0f;
00352     /*
00353     walkInterval_ is a float between 0.5 and 2.5,
00354     this is how it is generated:
00355
```

```
00356     rand() gives a number between 0 and RAND_MAX,
00357     so divide by RAND_MAX to get a float between 0 and 1,
00358     or in this case 0 and 2, because it is multiplied by 2,
00359     and then plus 0.5 of that to get float between 0.5 and 2.5.
00360     */
00361     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00362
00363     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00364     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00365     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00366     movement.x *= random_float_x * time;
00367     movement.y *= random_float_y * time;
00368     direction_ = movement;
00369
00370     if(rand() % 2) {
00371       sf::Vector2f shoot_direction1(10, 10);
00372       random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00373       random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00374       shoot_direction1.x *= random_float_x;
00375       shoot_direction1.y *= random_float_y;
00376       Projectile new_projectile1 = this->ShootAtDirection(shoot_direction1);
00377       active_projectiles.push_back(new_projectile1);
00378     }
00379     if(rand() % 2) {
00380       sf::Vector2f shoot_direction2(7, 7);
00381       sf::Vector2f direction_vector = player_position - body_.getPosition();
00382       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00383       shoot_direction2.x *= direction_vector.x / vector_magnitude;
00384       shoot_direction2.y *= direction_vector.y / vector_magnitude;
00385       Projectile new_projectile2 = this->ShootAtDirection(shoot_direction2);
00386       active_projectiles.push_back(new_projectile2);
00387     }
00388   }
00389   if(enemy_animations_[0].Update(time)) {
00390     body_.setTexture(enemy_animations_[0].getCurrentFrame());
00391   }
00392   body_.move(direction_);
00393     /*Collider col = this->getCollider();
00394     for (auto room : rooms) {
00395       for (auto wall : room.walls) {
00396         for (auto w : wall) {
00397           if (w.type_ == 1) {
00398             Collider wallCol = w.getCollider();
00399             wallCol.checkCollider(col, 1);
00400           }
00401         }
00402       }
00403     }*/
00404 };
```

5.5.4 Member Data Documentation

5.5.4.1 enemy_projectile_texture_

```
sf::Texture* FinalBoss::enemy_projectile_texture_
```

Definition at line 106 of file enemy.hpp.

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

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

5.6 HUD Class Reference

```
#include <HUD.hpp>
```

Public Member Functions

- HUD (sf::Font font, std::vector< sf::Texture * > hp, sf::Texture *background, sf::Texture *inventory, std::vector< sf::Texture * > item_textures)
- ~HUD ()
- void Display (sf::RenderWindow &window)
- void Update (std::vector< Item > &, float)

Public Attributes

- sf::RectangleShape background_
- std::vector< sf::RectangleShape > hp_bar_
- std::vector< sf::RectangleShape > inventory_bar_
- std::vector< sf::RectangleShape > hud_items_
- std::vector< sf::Text > texts_
- sf::Font font_
- std::vector< sf::Texture * > hp_textures_

5.6.1 Detailed Description

Definition at line 12 of file HUD.hpp.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 HUD()

```
HUD::HUD (
            sf::Font font,
            std::vector< sf::Texture * > hp,
            sf::Texture * background,
            sf::Texture * inventory,
            std::vector< sf::Texture * > item_textures )
```

Constructs HUD

Definition at line 6 of file HUD.cpp.

```
00009     : font_(font), hp_textures_(hp_textures) {
00010   // Window size 500x500 assumed
00011   background_.setSize(sf::Vector2f(500.0f, 150.0f));
00012   background_.setPosition(sf::Vector2f(0.0f, 500.0f));
00013   background_.setTexture(background);
00014   for (unsigned int i = 0; i < 3; i++) {
00015     sf::RectangleShape heart;
00016     heart.setSize(sf::Vector2f(50.0f, 50.0f));
00017     heart.setPosition(sf::Vector2f((i + 1) * 50.0f, 520.0f));
00018     heart.setTexture(hp_textures_[0]);
00019     hp_bar_.push_back(heart);
00020   }
00021   for (unsigned int i = 0; i < 3; i++) {
00022     sf::Text text;
00023     text.setString("0");
00024     text.setCharacterSize(24);
00025     text.setPosition(sf::Vector2f((i + 5) * 50.0f, 570.0f));
00026     texts_.push_back(text);
00027   }
```

```
00028   for (unsigned int i = 4; i < 7; i++) {
00029     sf::RectangleShape inv_slot;
00030     inv_slot.setSize(sf::Vector2f(50.0f, 50.0f));
00031     inv_slot.setPosition(sf::Vector2f((i + 1) * 50.0f, 520.0f));
00032     inv_slot.setTexture(inventory_texture);
00033     inventory_bar_.push_back(inv_slot);
00034   }
00035   for (unsigned int i = 0; i < item_textures.size(); i++) {
00036     sf::RectangleShape item_image;
00037     item_image.setSize(sf::Vector2f(50.0f, 50.0f));
00038     item_image.setPosition(sf::Vector2f((i + 5) * 50.0f, 520.0f));
00039     item_image.setTexture(item_textures[i]);
00040     hud_items_.push_back(item_image);
00041   }
00042 }
```

5.6.2.2 ~HUD()

```
HUD::~HUD ( )
```

Default destructor

Definition at line 126 of file HUD.cpp.

```
00126 {}
```

5.6.3 Member Function Documentation

5.6.3.1 Display()

```
void HUD::Display (
            sf::RenderWindow & window )
```

Rendrers the HUD

Definition at line 44 of file HUD.cpp.

```
00044                                       {
00045   window.draw(background_);
00046   for (unsigned int i = 0; i < hp_bar_.size(); i++) {
00047     window.draw(hp_bar_[i]);
00048   }
00049   for (unsigned int j = 0; j < inventory_bar_.size(); j++) {
00050     window.draw(inventory_bar_[j]);
00051     texts_[j].setFont(font_);
00052     window.draw(texts_[j]);
00053   }
00054   if (hud_items_.size() != 0) {
00055     for (unsigned int i = 0; i < hud_items_.size(); i++) {
00056       window.draw(hud_items_[i]);
00057     }
00058   }
00059 }
```

5.6.3.2 Update()

```
void HUD::Update (
            std::vector< Item > & items,
            float playerHP )
```

Updates the HUD

Definition at line 61 of file HUD.cpp.

```
00061                                                                {
00062   int hp_pots = 0;
00063   int spd_pots = 0;
00064   int coins = 0;
00065   for (auto i : items) {
00066     if (i.type == "hp_pot") {
00067       hp_pots++;
00068     }
00069     if (i.type == "speed_pot") {
00070       spd_pots++;
00071     }
00072     if (i.type == "coin") {
00073       coins++;
00074     }
00075   }
00076   std::stringstream ss;
00077   ss « hp_pots;
00078   texts_[0].setString(ss.str());
00079   std::stringstream ss2;
00080   ss2 « spd_pots;
00081   texts_[1].setString(ss2.str());
00082   std::stringstream ss3;
00083   ss3 « coins;
00084   texts_[2].setString(ss3.str());
00085
00086   int hp = static_cast<int>(playerHP);
00087   switch (hp) {
00088     case 6:
00089       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[0]);
00090       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00091       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00092       break;
00093     case 5:
00094       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[1]);
00095       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00096       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00097       break;
00098     case 4:
00099       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00100       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00101       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00102       break;
00103     case 3:
00104       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00105       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[1]);
00106       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00107       break;
00108     case 2:
00109       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00110       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00111       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00112       break;
00113     case 1:
00114       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00115       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00116       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[1]);
00117       break;
00118     case 0:
00119       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00120       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00121       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[2]);
00122       break;
00123   }
00124 }
```

5.6.4 Member Data Documentation

5.6.4.1 background_

```
sf::RectangleShape HUD::background_
```

Definition at line 23 of file HUD.hpp.

5.6.4.2 font_

```
sf::Font HUD::font_
```

Definition at line 28 of file HUD.hpp.

5.6.4.3 hp_bar_

```
std::vector<sf::RectangleShape> HUD::hp_bar_
```

Definition at line 24 of file HUD.hpp.

5.6.4.4 hp_textures_

```
std::vector<sf::Texture*> HUD::hp_textures_
```

Definition at line 29 of file HUD.hpp.

5.6.4.5 hud_items_

```
std::vector<sf::RectangleShape> HUD::hud_items_
```

Definition at line 26 of file HUD.hpp.

5.6.4.6 inventory_bar_

```
std::vector<sf::RectangleShape> HUD::inventory_bar_
```

Definition at line 25 of file HUD.hpp.

5.6.4.7 texts_

```
std::vector<sf::Text> HUD::texts_
```

Definition at line 27 of file HUD.hpp.

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

- src/HUD.hpp
- src/HUD.cpp

5.7 Item Class Reference

```
#include <item.hpp>
```

Public Member Functions

- Item (sf::Texture *texture, sf::Vector2f position, std::string type)
- ~Item ()
- void Draw (sf::RenderWindow &window)
- Collider getCollider ()

Public Attributes

- sf::RectangleShape body
- std::string type

5.7.1 Detailed Description

Class for handling items

Definition at line 10 of file item.hpp.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Item()

```
Item::Item (
            sf::Texture * texture,
            sf::Vector2f position,
            std::string type )
```

Constructor for Item oject

Definition at line 4 of file item.cpp.

```
00005     : type(type) {
00006   body.setSize(sf::Vector2f(30, 30));
00007   body.setPosition(position);
00008   body.setTexture(texture);
00009   body.setOrigin(sf::Vector2f(30, 30) / 2.0f);
00010 };
```

5.7.2.2 ~Item()

```
Item::~Item ( )  [inline]
```

Default destructor for item

Definition at line 15 of file item.hpp.
```
00015 {};
```

5.7.3 Member Function Documentation

5.7.3.1 Draw()

```
void Item::Draw (
            sf::RenderWindow & window )
```

Used for rendering the items

Definition at line 11 of file item.cpp.
```
00011 { window.draw(body); }
```

5.7.3.2 getCollider()

```
Collider Item::getCollider ( )  [inline]
```

Returns the item collider

Definition at line 19 of file item.hpp.
```
00019 { return Collider(body); }
```

5.7.4 Member Data Documentation

5.7.4.1 body

```
sf::RectangleShape Item::body
```

Definition at line 20 of file item.hpp.

5.7.4.2 type

```
std::string Item::type
```

Definition at line 21 of file item.hpp.

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

- src/item.hpp
- src/item.cpp

5.8 Map Class Reference

```
#include <map.hpp>
```

Public Member Functions

- Map (sf::Texture *wall_texture, Animation *enemy_animation, Animation *enemy_animation2, Animation *boss_animation, sf::Texture *enemy_projectile_texture)
- bool Generate (sf::Texture *wall_texture, Animation *enemy_animation, Animation *enemy_animation2, Animation *boss_animation, sf::Texture *enemy_projectile_texture, int x, int y, std::vector< int > openings, std::vector< std::pair< int, int >> &visited)
- void Display (sf::RenderWindow &window, Collider playerCollider)
- void NextRoom (int direction)
- void SpawnItem (unsigned int &i, sf::Texture *hp_text, sf::Texture *speed_text, sf::Texture *coin_text)
- void removeItem (std::vector< Item >::iterator i)

Public Attributes

- std::vector< Room > rooms
- std::vector< Enemy * > enemies
- std::vector< std::vector< Room > > layout
- sf::Vector2f mapSize = sf::Vector2f(6.0f, 6.0f)
- std::vector< Item > items

5.8.1 Detailed Description

Class for the map

Definition at line 19 of file map.hpp.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Map()

```
Map::Map (
            sf::Texture * wall_texture,
            Animation * enemy_animation,
            Animation * enemy_animation2,
            Animation * boss_animation,
            sf::Texture * enemy_projectile_texture )
```

Constructor for map

Definition at line 18 of file map.cpp.

```
00020                                                {
00021   std::vector<std::pair<int, int» visited;
00022   visited.push_back({100, 100});
00023   int depth = 0;
00024   Generate(wall_texture, enemy_animation, enemy_animation2, boss_animation,
00025            enemy_projectile_texture, 0, 0, {0, 1, 1, 0, 4}, visited);
00026
00027   int highest_depth = 0;
00028   unsigned int highest_depth_index = 0;
00029   for (unsigned int i = 0; i < rooms.size(); i++) {
00030     if (rooms[i].depth_ > highest_depth) {
00031       highest_depth = rooms[i].depth_;
00032       highest_depth_index = i;
00033     }
00034   }
00035   sf::Vector2f boss_spawn_pos(rooms[highest_depth_index].xBound1,
00036                               rooms[highest_depth_index].yBound1);
00037   sf::Vector2f enemy_spawn_pos;
00038   enemy_spawn_pos.x = boss_spawn_pos.x + 50 + (rand() % 350);
00039   enemy_spawn_pos.y = boss_spawn_pos.y + 50 + (rand() % 350);
00040
00041   FinalBoss *new_enemy = new FinalBoss(boss_animation, enemy_spawn_pos, 75,
00042                                        enemy_projectile_texture);
00043   // std::unique_ptr<Enemy> new_enemy(new FinalBoss(boss_animation,
00044   // enemy_spawn_pos, 75));
00045   enemies.push_back(new_enemy);
00046 };
```

5.8.3 Member Function Documentation

5.8.3.1 Display()

```
void Map::Display (
            sf::RenderWindow & window,
            Collider playerCollider )
```

Used for rendering the map

Definition at line 163 of file map.cpp.

```
00163                                                                {
00164   for (auto r : rooms) {
00165     r.Display(window, playerCollider, enemies);
00166   }
00167   for (auto i : items) {
00168     i.Draw(window);
00169   }
00170 };
```

5.8.3.2 Generate()

```
bool Map::Generate (
            sf::Texture * wall_texture,
            Animation * enemy_animation,
            Animation * enemy_animation2,
            Animation * boss_animation,
            sf::Texture * enemy_projectile_texture,
            int x,
            int y,
            std::vector< int > openings,
            std::vector< std::pair< int, int >> & visited )
```

Generates random dungeon and spawns random amount of enemies to each /*room.

Definition at line 48 of file map.cpp.

```
00052                                                                   {
00053   static int calls = 0;
00054   calls++;
00055   bool vi = false;
00056   for (auto m : visited) {
00057     if (x == m.first && y == m.second) {
00058       vi = true;
00059     }
00060   }
00061
00062   if (x < mapSize.x && y < mapSize.y && x >= 0 && y >= 0 &&
00063       std::accumulate(openings.begin(), openings.end() - 1, 0) > 1 &&
00064       vi == false) {
00065     visited.push_back({x, y});
00066     std::vector<int> new_openings;
00067     for (unsigned int i = 0; i < openings.size() - 1; i++) {
00068       if (openings[i] == 1 && openings[4] != int(i)) {
00069         new_openings.clear();
00070         for (auto j = 0; j < 4; j++) {
00071           new_openings.push_back(rand() % 2);
00072         }
00073         if (i < 2) {
00074           new_openings[i + 2] = 1;
00075           new_openings.push_back(i + 2);
00076         } else {
00077           new_openings[i - 2] = 1;
00078           new_openings.push_back(i - 2);
00079         }
00080         if (i == 0) {
00081           bool success = Generate(
00082               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00083               enemy_projectile_texture, x, y - 1, new_openings, visited);
00084           if (success == false) {
00085             openings[0] = 0;
00086           }
00087         } else if (i == 1) {
00088           bool success = Generate(
00089               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00090               enemy_projectile_texture, x + 1, y, new_openings, visited);
00091           if (success == false) {
00092             openings[1] = 0;
00093           }
00094         } else if (i == 2) {
00095           bool success = Generate(
00096               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00097               enemy_projectile_texture, x, y + 1, new_openings, visited);
00098           if (success == false) {
00099             openings[2] = 0;
00100           }
00101         } else {
00102           bool success = Generate(
00103               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00104               enemy_projectile_texture, x - 1, y, new_openings, visited);
00105           if (success == false) {
00106             openings[3] = 0;
00107           }
00108         }
00109       }
00110     }
00111
00112     /*
00113     The enemies are added inside the map and they are going to be located with
00114     respect to the room spawn positions. That means that every room is going to
```

```
00115     have a random amount of enemies located at random coordinates in the room,
00116     but the enemy location is not limited to the rooms, because enemies are
00117     located in the map, not in the rooms.
00118     */
00119     sf::Vector2f room_spawn_pos =
00120         sf::Vector2f(25.0f + 50 * 10 * x, 25.0f + 50 * 10 * y);
00121     // int random_enemy_amount = std::abs(rand() % 5);
00122     // int random_enemy_amount = rand() % 5;
00123     // int random_enemy_amount2 = rand() % 3;
00124     int random_enemy_amount = rand() % 4;
00125     int random_enemy_amount2 = rand() % 2;
00126     // int random_enemy_amount2 = 0;
00127     for (int i = 0; i < random_enemy_amount; i++) {
00128       sf::Vector2f enemy_spawn_pos;
00129       enemy_spawn_pos.x = room_spawn_pos.x + 50 + (rand() % 350);
00130       enemy_spawn_pos.y = room_spawn_pos.y + 50 + (rand() % 350);
00131       // std::unique_ptr<Enemy> new_enemy =
00132       // std::make_unique<ChasingEnemy>(enemy_animation, enemy_spawn_pos, 100);
00133       // ChasingEnemy new_enemy(enemy_animation, enemy_spawn_pos, 100);
00134       ChasingEnemy *new_enemy =
00135           new ChasingEnemy(enemy_animation, enemy_spawn_pos, 100);
00136       enemies.push_back(new_enemy);
00137     }
00138     for (int i = 0; i < random_enemy_amount2; i++) {
00139       sf::Vector2f enemy_spawn_pos;
00140       enemy_spawn_pos.x = room_spawn_pos.x + 50 + (rand() % 350);
00141       enemy_spawn_pos.y = room_spawn_pos.y + 50 + (rand() % 350);
00142       // std::unique_ptr<Enemy> new_enemy =
00143       // std::make_unique<ChasingEnemy>(enemy_animation, enemy_spawn_pos, 75);
00144       // std::unique_ptr<Enemy> new_enemy(new RangedEnemy(enemy_animation2,
00145       // enemy_spawn_pos, 75, enemy_projectile_texture)); RangedEnemy
00146       // new_enemy(enemy_animation2, enemy_spawn_pos, 75);
00147       RangedEnemy *new_enemy = new RangedEnemy(
00148           enemy_animation2, enemy_spawn_pos, 75, enemy_projectile_texture);
00149       enemies.push_back(new_enemy);
00150     }
00151
00152     Room room(wall_texture,
00153               sf::Vector2f(25.0f + 50 * 10 * x, 25.0f + 50 * 10 * y), openings,
00154               calls);
00155     rooms.push_back(room);
00156     calls--;
00157     return true;
00158   }
00159   calls--;
00160   return false;
00161 };
```

5.8.3.3 NextRoom()

```
void Map::NextRoom (
            int direction )
```

Used for whern player traverses between rooms.

Definition at line 172 of file map.cpp.

```
00172                                {
00173   for (unsigned int i = 0; i < rooms.size(); i++) {
00174     rooms[i].MoveRoom(direction, items);
00175   }
00176   for (unsigned int i = 0; i < enemies.size(); i++) {
00177     enemies[i]->MoveRoom(direction);
00178   }
00179 }
```

5.8.3.4 removeItem()

```
void Map::removeItem (
            std::vector< Item >::iterator i )
```

Used for picking up items

Definition at line 197 of file map.cpp.

```
00197 { items.erase(i); };
```

5.8.3.5 SpawnItem()

```
void Map::SpawnItem (
            unsigned int & i,
            sf::Texture * hp_text,
            sf::Texture * speed_text,
            sf::Texture * coin_text )
```

Used for spawning random item

Definition at line 181 of file map.cpp.

```
00182                                                                    {
00183   sf::Vector2f pos = enemies[i]->getPosition();
00184   int item_index = rand() % 3;
00185   if (item_index == 0) {
00186     Item drop(hp_text, pos, "hp_pot");
00187     items.push_back(drop);
00188   } else if (item_index == 1) {
00189     Item drop(speed_text, pos, "speed_pot");
00190     items.push_back(drop);
00191   } else {
00192     Item drop(coin_text, pos, "coin");
00193     items.push_back(drop);
00194   }
00195 }
```

5.8.4 Member Data Documentation

5.8.4.1 enemies

```
std::vector<Enemy*> Map::enemies
```

Definition at line 45 of file map.hpp.

5.8.4.2 items

```
std::vector<Item> Map::items
```

Definition at line 48 of file map.hpp.

5.8.4.3 layout

```
std::vector<std::vector<Room> > Map::layout
```

Definition at line 46 of file map.hpp.

5.8.4.4 mapSize

```
sf::Vector2f Map::mapSize = sf::Vector2f(6.0f, 6.0f)
```

Definition at line 47 of file map.hpp.

5.8.4.5 rooms

```
std::vector<Room> Map::rooms
```

Definition at line 42 of file map.hpp.

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

- src/map.hpp
- src/map.cpp

5.9 Menu Class Reference

```
#include <menu.hpp>
```

Public Member Functions

- Menu (sf::Font font, sf::Texture *menubackground)
- ~Menu ()
- void Display (sf::RenderWindow &window)
- void Update (sf::RenderWindow &window)

Public Attributes

- sf::RectangleShape background_
- sf::Font font_
- std::vector< sf::Text > texts_
- bool active_

5.9.1 Detailed Description

Class for main menu

Definition at line 13 of file menu.hpp.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Menu()

```
Menu::Menu (
            sf::Font font,
            sf::Texture * menubackground )
```

Constructor for menu

Definition at line 11 of file menu.cpp.

```
00012     : font_(font), active_(true) {
00013   background_.setSize(sf::Vector2f(500, 650));
00014   background_.setPosition(sf::Vector2f(0, 0));
00015   background_.setTexture(menubackground);
00016   sf::Text StartText;
00017   StartText.setString("Start New Game");
00018   StartText.setCharacterSize(36);
00019   StartText.setPosition(50, 300);
00020   StartText.setFillColor(sf::Color::Yellow);
00021   StartText.setFont(font_);
00022   texts_.push_back(StartText);
00023   sf::Text QuitText;
00024   QuitText.setString("Quit Game");
00025   QuitText.setCharacterSize(36);
00026   QuitText.setPosition(50, 350);
00027   QuitText.setFillColor(sf::Color::White);
00028   QuitText.setFont(font_);
00029   texts_.push_back(QuitText);
00030 }
```

5.9.2.2 ~Menu()

```
Menu::~Menu ( )
```

Default destructor

Definition at line 40 of file menu.cpp.

```
00040 {};
```

5.9.3 Member Function Documentation

5.9.3.1 Display()

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

Renders the main menu

Definition at line 31 of file menu.cpp.

```
00031                                       {
00032   if (active_) {
00033     this->Update(window);
00034     window.draw(background_);
00035     for (unsigned int i = 0; i < texts_.size(); i++) {
00036       window.draw(texts_[i]);
00037     }
00038   }
00039 }
```

5.9.3.2 Update()

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

Polls for actions in main menu

Definition at line 42 of file menu.cpp.

```
00042                                           {
00043   if (sf::Mouse::isButtonPressed(sf::Mouse::Left) and active_) {
00044     sf::Vector2i pos = sf::Mouse::getPosition(relativeTo);
00045     if (pos.y > 30 && pos.y < (300 + 36)) {
00046       active_ = false;
00047     }
00048     if (pos.y > (350) && pos.y < (350 + 36)) {
00049       relativeTo.close();
00050     }
00051   }
00052 }
```

5.9.4 Member Data Documentation

5.9.4.1 active_

```
bool Menu::active_
```

Definition at line 26 of file menu.hpp.

5.9.4.2 background_

```
sf::RectangleShape Menu::background_
```

Definition at line 23 of file menu.hpp.

5.9.4.3 font_

```
sf::Font Menu::font_
```

Definition at line 24 of file menu.hpp.

5.9.4.4 texts_

```
std::vector<sf::Text> Menu::texts_
```

Definition at line 25 of file menu.hpp.

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

- src/menu.hpp
- src/menu.cpp

5.10 Player Class Reference

```
#include <player.hpp>
```

Public Member Functions

- Player (sf::Texture *texture, sf::Texture *dead_texture, sf::Vector2f spawnPos, float speed, Weapon *starting_weapon)
- ~Player ()
- void Draw (sf::RenderWindow &window)
- sf::Vector2f getPosition ()
- bool isAlive ()
- float getHp ()
- float getShield ()
- bool useItem ()
- void Damage (float dmg)
- std::vector< int > Update (float time, Map &map, std::vector< Projectile > &)
- Projectile UseWeapon (sf::Vector2f &)
- void Traverse (Map &map)
- Collider getCollider ()

Public Attributes

- float speed_
- float hp_ = 6
- float shield_ = 0
- float currentSpeed_
- float actionTimer_
- float shot_ = 0.0
- float damaged_ = 0.0
- int chosen_weapon_ = 0
- float speedTimer = 0
- float itemTimer = 0
- bool alive_ = true
- bool dead_ = false
- sf::Texture * death_texture_
- std::vector< Weapon > player_weapons_
- std::vector< Item > items_
- sf::RectangleShape body_
- sf::Vector2f direction_

5.10.1 Detailed Description

Class for the player object

Definition at line 15 of file player.hpp.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Player()

```
Player::Player (
            sf::Texture * texture,
            sf::Texture * dead_texture,
            sf::Vector2f spawnPos,
            float speed,
            Weapon * starting_weapon )
```

Constructor for player

Definition at line 7 of file player.cpp.

```
00009     : speed_(speed) {
00010   death_texture_ = dead_texture;
00011   player_weapons_.push_back(*starting_weapon);
00012   body_.setSize(sf::Vector2f(40.0f, 40.0f));
00013   body_.setOrigin(body_.getSize() / 2.0f);
00014   body_.setPosition(spawnPos);
00015   body_.setTexture(texture);
00016 };
```

5.10.2.2 ~Player()

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

Default constructor

Definition at line 18 of file player.cpp.

```
00018 {};
```

5.10.3 Member Function Documentation

5.10.3.1 Damage()

```
void Player::Damage (
            float dmg )
```

Decrease health

Definition at line 65 of file player.cpp.

```
00065                                {
00066   if (shield_ > 0) {
00067     shield_ -= damage;
00068     if (shield_ < 0) shield_ = 0;
00069   } else
00070     hp_ -= damage;
00071   if (hp_ <= 0) {
00072     std::cout « "The player has died!" « std::endl;
00073     alive_ = false;
00074   }
00075 }
```

5.10.3.2 Draw()

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

Renders the player

Definition at line 20 of file player.cpp.

```
00020 { window.draw(body_); };
```

5.10.3.3 getCollider()

```
Collider Player::getCollider ( )  [inline]
```

Returns the player collider

Definition at line 45 of file player.hpp.

```
00045 { return Collider(body_); }
```

5.10.3.4 getHp()

```
float Player::getHp ( )
```

Return healh

Definition at line 26 of file player.cpp.

```
00026 { return hp_; }
```

5.10.3.5 getPosition()

```
sf::Vector2f Player::getPosition ( )
```

Returns player position

Definition at line 22 of file player.cpp.

```
00022 { return body_.getPosition(); };
```

5.10.3.6 getShield()

```
float Player::getShield ( )
```

Definition at line 28 of file player.cpp.

```
00028 { return shield_; }
```

5.10.3.7 isAlive()

```
bool Player::isAlive ( )
```

Return alive

Definition at line 24 of file player.cpp.

```
00024 { return alive_; };
```

5.10.3.8 Traverse()

```
void Player::Traverse (
            Map & map )
```

Moves the map when player traverses between rooms

Definition at line 195 of file player.cpp.

```
00195                                {
00196   sf::Vector2f pos = body_.getPosition();
00197
00198   if (pos.x < 25) {
00199     map.NextRoom(1);
00200     body_.move(sf::Vector2f(500, 0));
00201   }
00202   if (pos.x > 25 + 10 * 50) {
00203     map.NextRoom(3);
00204     body_.move(sf::Vector2f(-500, 0));
00205   }
00206   if (pos.y < 25) {
00207     map.NextRoom(2);
00208     body_.move(sf::Vector2f(0, 500));
00209   }
00210   if (pos.y > 25 + 10 * 50) {
00211     map.NextRoom(0);
00212     body_.move(sf::Vector2f(0, -500));
00213   }
00214 }
```

5.10.3.9 Update()

```
std::vector< int > Player::Update (
            float time,
            Map & map,
            std::vector< Projectile > & active_projectiles )
```

Update attributes

Definition at line 77 of file player.cpp.

```
00078                                                                                {
00079   std::vector<int> ret = {0, 0, 0, 0};
00080   if (alive_ == false) {
00081     if (dead_ == false) {
00082       sf::Vector2u texture_size = death_texture_->getSize();
00083       sf::Vector2f texture_size_f(float(texture_size.x), float(texture_size.y));
00084       body_.setSize(texture_size_f);
00085       body_.setOrigin(body_.getSize() / 2.0f);
00086       sf::Vector2f temp_position = body_.getPosition();
00087       std::cout « "Death position: " « temp_position.x « " "
00088                 « temp_position.y « std::endl;
00089       body_.setPosition(temp_position);
00090       body_.setTexture(death_texture_);
00091       dead_ = true;
00092     }
00093     // return active_projectiles;
00094   }
00095   if (this->useItem()) {
00096     ret[2] = 1;
00097   }
00098   currentSpeed_ = speed_;
00099   actionTimer_ += time;
00100   shot_ += time;  // cooldown for firing weapon
00101   itemTimer += time;
00102   sf::Vector2f movement(0.0f, 0.0f);
00103   float firerate = player_weapons_[chosen_weapon_].fire_rate_;
00104
00105   if (speedTimer > 0) {
00106     speedTimer -= time;
00107   } else if (speedTimer < 0) {
00108     speedTimer = 0;
00109     speed_ -= 10;
00110   }
00111
00112   if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
00113     movement.y -= currentSpeed_ * time;
00114     direction_ = movement;
00115   }
00116
00117   if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
00118     movement.x -= currentSpeed_ * time;
00119     direction_ = movement;
00120   }
00121
00122   if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
00123     movement.y += currentSpeed_ * time;
00124     direction_ = movement;
00125   }
00126
00127   if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
00128     movement.x += currentSpeed_ * time;
00129     direction_ = movement;
00130   }
00131
00132   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) &&
00133       shot_ >= firerate) {  // Up arrow key is used to shoot projectiles.
00134     sf::Vector2f direction = sf::Vector2f(0.0f, -10.0f);
00135     Projectile new_projectile = this->UseWeapon(direction);
00136     active_projectiles.push_back(new_projectile);
00137     shot_ = 0.0;
00138     ret[0] = 1;
00139   }
00140   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) &&
00141       shot_ >= firerate) {  // Down arrow key is used to shoot projectiles.
00142     sf::Vector2f direction = sf::Vector2f(0.0f, 10.0f);
00143     Projectile new_projectile = this->UseWeapon(direction);
00144     active_projectiles.push_back(new_projectile);
00145     shot_ = 0.0;
00146     ret[0] = 1;
00147   }
00148   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) &&
00149       shot_ >= firerate) {  // Left arrow key is used to shoot projectiles.
```

```
00150     sf::Vector2f direction = sf::Vector2f(-10.0f, 0.0f);
00151     Projectile new_projectile = this->UseWeapon(direction);
00152     active_projectiles.push_back(new_projectile);
00153     shot_ = 0.0;
00154     ret[0] = 1;
00155   }
00156   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) &&
00157       shot_ >= firerate) {  // Right arrow key is used to shoot projectiles.
00158     sf::Vector2f direction = sf::Vector2f(10.0f, 0.0f);
00159     Projectile new_projectile = this->UseWeapon(direction);
00160     active_projectiles.push_back(new_projectile);
00161     shot_ = 0.0;
00162     ret[0] = 1;
00163   }
00164
00165   Traverse(map);
00166   body_.move(movement);
00167
00168   // check that player is not colliding with any of the enemies and take damage
00169   // if colliding
00170   damaged_ += time;
00171   Collider collider = this->getCollider();
00172   for (auto& enemy : map.enemies) {
00173     Collider EnemyCollider = enemy->getCollider();
00174     if (collider.checkCollider(EnemyCollider, 0.5) && damaged_ >= 1.5) {
00175       this->Damage(enemy->damage_);
00176       damaged_ = 0.0;
00177       ret[1] = 1;
00178     }
00179   }
00180   for (unsigned int i = 0; i < map.items.size(); i++) {
00181     Collider itemCollider = map.items[i].getCollider();
00182     if (collider.checkCollider(itemCollider, 0)) {
00183       items_.push_back(map.items[i]);
00184       map.items.erase(map.items.begin() + i);
00185       ret[3] = 1;
00186     }
00187   }
00188   return ret;
00189 };
```

5.10.3.10 useItem()

```
bool Player::useItem ( )
```

Uses item from inventory

Definition at line 30 of file player.cpp.

```
00030                       {
00031   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num1)) {
00032     auto i = items_.begin();
00033     while (i != items_.end()) {
00034       if (i->type == "hp_pot" && itemTimer > 0.5) {
00035         if (hp_ < 6) {
00036           hp_ += 1;
00037           i = items_.erase(i);
00038           itemTimer = 0;
00039           return true;
00040         } else {
00041           return false;
00042         }
00043       } else {
00044         ++i;
00045       }
00046     }
00047   }
00048
00049   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num2)) {
00050     auto i = items_.begin();
00051     while (i != items_.end()) {
00052       if (i->type == "speed_pot" && itemTimer > 0.5) {
00053         speed_ += 10;
00054         speedTimer = 15;
00055         i = items_.erase(i);
00056         itemTimer = 0;
00057         return true;
00058       } else {
00059         ++i;
```

```
00060       }
00061     }
00062   }
00063 }
```

5.10.3.11 UseWeapon()

```
Projectile Player::UseWeapon (
            sf::Vector2f & direction )
```

Creates projectile

Definition at line 191 of file player.cpp.

```
00191                                                 {
00192   return player_weapons_[chosen_weapon_].Fire(body_.getPosition(), direction);
00193 };
```

5.10.4 Member Data Documentation

5.10.4.1 actionTimer_

```
float Player::actionTimer_
```

Definition at line 51 of file player.hpp.

5.10.4.2 alive_

```
bool Player::alive_ = true
```

Definition at line 57 of file player.hpp.

5.10.4.3 body_

```
sf::RectangleShape Player::body_
```

Definition at line 62 of file player.hpp.

5.10.4.4 chosen_weapon_

```
int Player::chosen_weapon_ = 0
```

Definition at line 54 of file player.hpp.

5.10.4.5 currentSpeed_

```
float Player::currentSpeed_
```

Definition at line 50 of file player.hpp.

5.10.4.6 damaged_

```
float Player::damaged_ = 0.0
```

Definition at line 53 of file player.hpp.

5.10.4.7 dead_

```
bool Player::dead_ = false
```

Definition at line 58 of file player.hpp.

5.10.4.8 death_texture_

```
sf::Texture* Player::death_texture_
```

Definition at line 59 of file player.hpp.

5.10.4.9 direction_

```
sf::Vector2f Player::direction_
```

Definition at line 63 of file player.hpp.

5.10.4.10 hp_

```
float Player::hp_ = 6
```

Definition at line 48 of file player.hpp.

5.10.4.11 items_

```
std::vector<Item> Player::items_
```

Definition at line 61 of file player.hpp.

5.10.4.12 itemTimer

```
float Player::itemTimer = 0
```

Definition at line 56 of file player.hpp.

5.10.4.13 player_weapons_

```
std::vector<Weapon> Player::player_weapons_
```

Definition at line 60 of file player.hpp.

5.10.4.14 shield_

```
float Player::shield_ = 0
```

Definition at line 49 of file player.hpp.

5.10.4.15 shot_

```
float Player::shot_ = 0.0
```

Definition at line 52 of file player.hpp.

5.10.4.16 speed_

```
float Player::speed_
```

Definition at line 47 of file player.hpp.

5.10.4.17 speedTimer

```
float Player::speedTimer = 0
```

Definition at line 55 of file player.hpp.

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

- src/player.hpp
- src/player.cpp

5.11 Projectile Class Reference

```
#include <projectile.hpp>
```

Public Member Functions

- Projectile (sf::Texture *texture)
- ~Projectile ()
- void Draw (sf::RenderWindow &window)
- bool Update (float time, Map &map, Player &player)
- void activate ()
- void deActivate ()
- Collider getCollider ()

Public Attributes

- sf::Texture * projectile_texture_
- sf::RectangleShape body_
- sf::Vector2f projectile_trajectory_
- bool active_ = false
- float actionTimer_ = 0.0

5.11.1 Detailed Description

Class for projectiles

Definition at line 14 of file projectile.hpp.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Projectile()

```
Projectile::Projectile (
            sf::Texture * texture )
```

Constructor

Definition at line 9 of file projectile.cpp.

```
00009                                          {
00010   //friendly_ = true;
00011   //friendly_.push_back(1);
00012   projectile_texture_ = texture;
00013 };
```

5.11.2.2 ~Projectile()

```
Projectile::~Projectile ( )
```

Default destructor

Definition at line 18 of file projectile.cpp.

```
00018 {};
```

5.11.3 Member Function Documentation

5.11.3.1 activate()

```
void Projectile::activate ( )
```

Changes active_ to true

Definition at line 76 of file projectile.cpp.

```
00076 {active_ = true;};
```

5.11.3.2 deActivate()

```
void Projectile::deActivate ( )
```

Changes active_ to false

Definition at line 77 of file projectile.cpp.

```
00077 {active_ = false;};
```

5.11.3.3 Draw()

```
void Projectile::Draw (
            sf::RenderWindow & window )
```

Renders the projectile

Definition at line 23 of file projectile.cpp.

```
00023 { window.draw(body_); };
```

5.11.3.4 getCollider()

```
Collider Projectile::getCollider ( )  [inline]
```

Returns the projectile collider

Definition at line 29 of file projectile.hpp.

```
00029                          {
00030     return Collider(body_);
00031   }
```

5.11.3.5 Update()

```
bool Projectile::Update (
            float time,
            Map & map,
            Player & player )
```

Updates attributes

Definition at line 29 of file projectile.cpp.

```
00029                                                                 {
00030   actionTimer_ += time;
00031   if(actionTimer_ > 1.0) {
00032     return 1;
00033   }
00034   body_.move(projectile_trajectory_);
00035   Collider col = this->getCollider();
00036   //std::cout « friendly_;
00037   //if(friendly_ == 1) {
00038
00039     //secret_size is used for determining who the projectile harms, because otherwise there will be
      memory issues.
00040   sf::Vector2u secret_size = this->projectile_texture_->getSize();
00041   if(secret_size.x < 500) { //Enemy projectile textures happen to be, by coincidence, larger than
      player projectile textures
00042     for (auto& enemy : map.enemies) {
00043     Collider enemyCollider = enemy->getCollider();
00044     if (col.checkCollider(enemyCollider, 0)) {
00045         enemy->Damage(15);
00046         return 1;
00047     }
00048     }
00049   } else {
00050     Collider playerCollider = player.getCollider();
00051     if (col.checkCollider(playerCollider, 0)) {
00052         player.Damage(2);
00053         return 1;
00054     }
00055   }
00056   for (auto room : map.rooms) {
00057     for (auto wall : room.walls) {
00058       for (auto w : wall) {
```

```
00059         if (w.type_ == 1) {
00060           Collider wallCollider = w.getCollider();
00061           if (wallCollider.checkCollider(col, 1)) {
00062             return 1;
00063           }
00064         }
00065       }
00066     }
00067   }
00068   return 0;
00069   //std::cout « projectile_trajectory_.x « "\t" « projectile_trajectory_.y « "\n";
00070   //std::cout « body_.getPosition().x « "\t" « body_.getPosition().y « "\n";
00071 };
```

5.11.4 Member Data Documentation

5.11.4.1 actionTimer_

```
float Projectile::actionTimer_ = 0.0
```

Definition at line 37 of file projectile.hpp.

5.11.4.2 active_

```
bool Projectile::active_ = false
```

Definition at line 36 of file projectile.hpp.

5.11.4.3 body_

```
sf::RectangleShape Projectile::body_
```

Definition at line 34 of file projectile.hpp.

5.11.4.4 projectile_texture_

```
sf::Texture* Projectile::projectile_texture_
```

Definition at line 33 of file projectile.hpp.

5.11.4.5 projectile_trajectory_

```
sf::Vector2f Projectile::projectile_trajectory_
```

Definition at line 35 of file projectile.hpp.

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

- src/projectile.hpp
- src/projectile.cpp

5.12 RangedEnemy Class Reference

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

Inheritance diagram for RangedEnemy:

Collaboration diagram for RangedEnemy:

Public Member Functions

- RangedEnemy (Animation *enemy_animation, sf::Vector2f spawnPos, float speed, sf::Texture *enemy_↩ projectile_texture)
- ~RangedEnemy ()
- virtual void Update (float time, sf::Vector2f player_position) override
- void Update2 (float time, sf::Vector2f player_position, std::vector< Projectile > &active_projectiles)
- Projectile ShootAtDirection (sf::Vector2f direction)

Public Attributes

- sf::Texture * enemy_projectile_texture_

5.12.1 Detailed Description

Class for enemy that shoots projectiles

Definition at line 75 of file enemy.hpp.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 RangedEnemy()

```
RangedEnemy::RangedEnemy (
            Animation * enemy_animation,
            sf::Vector2f spawnPos,
            float speed,
            sf::Texture * enemy_projectile_texture )
```

Definition at line 128 of file enemy.cpp.

```
00130   : Enemy(enemy_animation, spawnPos, speed), enemy_projectile_texture_(enemy_projectile_texture)/*,
        is_ranged_type_(true)*/ {
00131       is_ranged_type_ = true;
00132       //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00133   };
```

5.12.2.2 ~RangedEnemy()

```
RangedEnemy::~RangedEnemy ( )
```

Definition at line 135 of file enemy.cpp.

```
00135 {};
```

5.12.3 Member Function Documentation

5.12.3.1 ShootAtDirection()

```
Projectile RangedEnemy::ShootAtDirection (
            sf::Vector2f direction )
```

Used to shoot projetiles towards direction

Definition at line 259 of file enemy.cpp.

```
00259                                                       {
00260   //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00261   //Projectile Weapon::Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory)
00262   Projectile proj(this->enemy_projectile_texture_);
00263   //proj.friendly_[0] = 0;
00264   proj.active_ = true;
00265   proj.body_.setTexture(proj.projectile_texture_);
00266   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00267   proj.body_.setPosition(body_.getPosition());
00268   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00269   proj.projectile_trajectory_ = direction/* * proj.velocity_multiplier_*/;
00270   proj.body_.move(proj.projectile_trajectory_);
00271   return proj;
00272 };
```

5.12.3.2 Update()

```
void RangedEnemy::Update (
            float time,
            sf::Vector2f player_position )  [override], [virtual]
```

Virtual function for updating enemy position

Implements Enemy.

Definition at line 137 of file enemy.cpp.

```
00137                                                                   {
00138   currentSpeed_ = speed_;
00139   actionTimer_ += time;
00140
00141   /* Change walk direction to some random direction at random intervals */
00142
00143   if(actionTimer_ >= walkInterval_){
00144     actionTimer_ = 0.0f;
00145     walk_ = walk_ ? false : true; //tenary operator
00146
00147     /*
00148     walkInterval_ is a float between 0.5 and 2.5,
00149     this is how it is generated:
00150
00151     rand() gives a number between 0 and RAND_MAX,
00152     so divide by RAND_MAX to get a float between 0 and 1,
00153     or in this case 0 and 2, because it is multiplied by 2,
00154     and then plus 0.5 of that to get float between 0.5 and 2.5.
00155     */
00156     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00157
00158     if(walk_ == false){
00159       enemy_animations_[0].resetCurrentFrame();
00160       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00161     }
00162
00163     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00164     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00165     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00166     movement.x *= random_float_x * time;
00167     movement.y *= random_float_y * time;
00168     direction_ = movement;
00169   }
00170
00171   if(walk_){
00172     if(enemy_animations_[0].Update(time)){
00173       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00174     }
00175     body_.move(direction_);
00176     /*Collider col = this->getCollider();
00177     for (auto room : rooms) {
00178       for (auto wall : room.walls) {
00179         for (auto w : wall) {
00180           if (w.type_ == 1) {
00181             Collider wallCol = w.getCollider();
00182             wallCol.checkCollider(col, 1);
00183           }
00184         }
00185       }
00186     }*/
00187   }
00188 };
```

5.12.3.3 Update2()

```
void RangedEnemy::Update2 (
            float time,
            sf::Vector2f player_position,
            std::vector< Projectile > & active_projectiles )  [virtual]
```

Reimplemented from Enemy.

Definition at line 190 of file enemy.cpp.

```
00190
        {
00191   currentSpeed_ = speed_;
00192   actionTimer_ += time;
00193
00194   /* Change walk direction to some random direction at random intervals */
00195
00196   if(actionTimer_ >= walkInterval_){
00197     actionTimer_ = 0.0f;
00198     walk_ = walk_ ? false : true; //tenary operator
00199
00200     /*
00201     walkInterval_ is a float between 0.5 and 2.5,
00202     this is how it is generated:
00203
00204     rand() gives a number between 0 and RAND_MAX,
00205     so divide by RAND_MAX to get a float between 0 and 1,
00206     or in this case 0 and 2, because it is multiplied by 2,
00207     and then plus 0.5 of that to get float between 0.5 and 2.5.
00208     */
00209     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00210
00211     if(walk_ == false){
00212       enemy_animations_[0].resetCurrentFrame();
00213       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00214     }
00215
00216     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00217     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00218     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00219     movement.x *= random_float_x * time;
00220     movement.y *= random_float_y * time;
00221     direction_ = movement;
00222
00223     if(rand() % 2) { //Shoot somewhere: 50% chance
00224       sf::Vector2f shoot_direction(10, 10);
00225       if(rand() % 4 == 1) { //Shoot at random direction: 25% chance
00226         random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00227         random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00228         shoot_direction.x *= random_float_x;
00229         shoot_direction.y *= random_float_y;
00230       } else { //Shoot at direction of player: 75% chance
00231         sf::Vector2f direction_vector = player_position - body_.getPosition();
00232         float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00233         shoot_direction.x *= direction_vector.x / vector_magnitude;
00234         shoot_direction.y *= direction_vector.y / vector_magnitude;
00235       }
00236       Projectile new_projectile = this->ShootAtDirection(shoot_direction);
00237       active_projectiles.push_back(new_projectile);
00238     }
00239   }
00240   if(walk_){
00241     if(enemy_animations_[0].Update(time)){
00242       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00243     }
00244     body_.move(direction_);
00245     /*Collider col = this->getCollider();
00246     for (auto room : rooms) {
00247       for (auto wall : room.walls) {
00248         for (auto w : wall) {
00249           if (w.type_ == 1) {
00250             Collider wallCol = w.getCollider();
00251             wallCol.checkCollider(col, 1);
00252           }
00253         }
00254       }
00255     }*/
00256   }
00257 };
```

5.12.4 Member Data Documentation

5.12.4.1 enemy_projectile_texture_

```
sf::Texture* RangedEnemy::enemy_projectile_texture_
```

Definition at line 88 of file enemy.hpp.

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

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

5.13 Room Class Reference

`#include <room.hpp>`

Public Member Functions

- Room (sf::Texture *texture, sf::Vector2f spawnPos, std::vector< int > openings, int depth)
- void Display (sf::RenderWindow &window, Collider playerCollider, std::vector< Enemy * > &enemies)
- void checkCollision (sf::RectangleShape &other_body)
- void MoveRoom (int direction, std::vector< Item > &items)
- ~Room ()
- void Activate ()
- void Deactivate ()

Public Attributes

- bool active
- sf::Vector2f maxSize = sf::Vector2f(10.0f, 10.0f)
- float xBound1
- float xBound2
- float yBound1
- float yBound2
- std::vector< std::vector< Wall > > walls
- int depth_

5.13.1 Detailed Description

Definition at line 14 of file room.hpp.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Room()

```
Room::Room (
            sf::Texture * texture,
            sf::Vector2f spawnPos,
            std::vector< int > openings,
            int depth )
```

Definition at line 11 of file room.cpp.

```
00013     : active(false), depth_(depth) {
00014   xBound1 = spawnPos.x;
00015   xBound2 = spawnPos.x + maxSize.x * 50.0f;
00016
00017   yBound1 = spawnPos.y;
00018   yBound2 = spawnPos.y + maxSize.y * 50.0f;
00019
00020   sf::Texture *floortexture = new sf::Texture();
00021   floortexture->loadFromFile("libs/images/floor_1.png");
00022
00023   for (int x = 0; x < maxSize.x; x++) {
00024     walls.push_back(std::vector<Wall>());
00025
00026     for (int y = 0; y < maxSize.y; y++) {
00027       if (y == floor(maxSize.y / 2) &&
00028           ((x == 0 && openings[3] == 1) ||
00029            (x == maxSize.x - 1 && openings[1] == 1))) {
00030         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00031                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 2);
00032         walls[x].push_back(wall);
00033       } else if (x == floor(maxSize.x / 2) &&
00034                  ((y == 0 && openings[0] == 1) ||
00035                   (y == maxSize.y - 1 && openings[2] == 1))) {
00036         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00037                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 2);
00038         walls[x].push_back(wall);
00039       }
00040
00041       else if (x == 0 || x == maxSize.x - 1 || y == 0 || y == maxSize.y - 1) {
00042         Wall wall(texture, sf::Vector2f(50.0f, 50.f),
00043                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 1);
00044         walls[x].push_back(wall);
00045
00046       } else {
00047         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00048                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 0);
00049         walls[x].push_back(wall);
00050       }
00051     }
00052   }
00053 };
```

5.13.2.2 ~Room()

```
Room::~Room ( )  [inline]
```

Default destructor

Definition at line 32 of file room.hpp.

```
00032 {};
```

5.13.3 Member Function Documentation

5.13.3.1 Activate()

```
void Room::Activate ( )
```

Definition at line 122 of file room.cpp.

```
00122 { active = true; };
```

5.13.3.2 checkCollision()

```
void Room::checkCollision (
            sf::RectangleShape & other_body )
```

Checks collisions with walls

5.13.3.3 Deactivate()

```
void Room::Deactivate ( )
```

Definition at line 123 of file room.cpp.

```
00123 { active = false; }
```

5.13.3.4 Display()

```
void Room::Display (
            sf::RenderWindow & window,
            Collider playerCollider,
            std::vector< Enemy * > & enemies )
```

Definition at line 59 of file room.cpp.

```
00060                                                {
00061   for (auto r : walls) {
00062     for (auto w : r) {
00063       w.Draw(window);
00064       Collider wallCollider = w.getCollider();
00065       if (w.type_ == 1) {
00066         wallCollider.checkCollider(playerCollider, 1);
00067
00068         for (auto i = 0; i < enemies.size(); i++) {
00069           Collider enemyCollider = enemies[i]->getCollider();
00070           wallCollider.checkCollider(enemyCollider, 1);
00071         }
00072       }
00073       if (w.type_ == 2) {
00074         for (auto i = 0; i < enemies.size(); i++) {
00075           Collider enemyCollider = enemies[i]->getCollider();
00076           wallCollider.checkCollider(enemyCollider, 1);
00077         }
00078       }
00079     }
00080   }
00081 };
```

5.13.3.5 MoveRoom()

```
void Room::MoveRoom (
            int direction,
            std::vector< Item > & items )
```

Moves the room when player traverses

Definition at line 97 of file room.cpp.

```
00097                                                                    {
00098   sf::Vector2f dir;
00099   if (direction == 0) {
00100     dir = sf::Vector2f(0, -10 * 50);
00101   }
00102   if (direction == 1) {
00103     dir = sf::Vector2f(10 * 50, 0);
00104   }
00105   if (direction == 2) {
00106     dir = sf::Vector2f(0, 10 * 50);
00107   }
00108   if (direction == 3) {
00109     dir = sf::Vector2f(-10 * 50, 0);
00110   }
00111
00112   for (auto i = 0; i < walls.size(); i++) {
00113     for (auto j = 0; j < walls[i].size(); j++) {
00114       walls[i][j].Move2(dir);
00115     }
00116   }
00117   for (auto i = 0; i < items.size(); i++) {
00118     items[i].body.move(dir);
00119   }
00120 }
```

5.13.4 Member Data Documentation

5.13.4.1 active

```
bool Room::active
```

Definition at line 35 of file room.hpp.

5.13.4.2 depth_

```
int Room::depth_
```

Definition at line 42 of file room.hpp.

5.13.4.3 maxSize

```
sf::Vector2f Room::maxSize = sf::Vector2f(10.0f, 10.0f)
```

Definition at line 36 of file room.hpp.

5.13.4.4 walls

```
std::vector<std::vector<Wall> > Room::walls
```

Definition at line 41 of file room.hpp.

5.13.4.5 xBound1

```
float Room::xBound1
```

Definition at line 37 of file room.hpp.

5.13.4.6 xBound2

```
float Room::xBound2
```

Definition at line 38 of file room.hpp.

5.13.4.7 yBound1

```
float Room::yBound1
```

Definition at line 39 of file room.hpp.

5.13.4.8 yBound2

```
float Room::yBound2
```

Definition at line 40 of file room.hpp.

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

- src/room.hpp
- src/room.cpp

5.14 Wall Class Reference

```
#include <wall.hpp>
```

Public Member Functions

- Wall (sf::Texture *texture, sf::Vector2f size, sf::Vector2f position, int type)
- void Draw (sf::RenderWindow &window)
- void Move2 (sf::Vector2f dir)
- bool checkCollision (sf::RectangleShape other_body)
- Collider getCollider ()

Public Attributes

- sf::RectangleShape body
- int type_

5.14.1 Detailed Description

Class for walls

Definition at line 11 of file wall.hpp.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Wall()

```
Wall::Wall (
            sf::Texture * texture,
            sf::Vector2f size,
            sf::Vector2f position,
            int type )
```

Constructor

Definition at line 5 of file wall.cpp.

```
00006                     {
00007   body.setSize(size);
00008   body.setPosition(position);
00009   body.setTexture(texture);
00010   body.setOrigin(size / 2.0f);
00011   type_ = type;
00012 };
```

5.14.3 Member Function Documentation

5.14.3.1 checkCollision()

```
bool Wall::checkCollision (
            sf::RectangleShape other_body )
```

Checks for collisions

Definition at line 17 of file wall.cpp.

```
00017                                                    {
00018   return body.getGlobalBounds().intersects(other_body.getGlobalBounds());
00019 }
```

5.14.3.2 Draw()

```
void Wall::Draw (
            sf::RenderWindow & window )
```

Renders the wall

Definition at line 13 of file wall.cpp.

```
00013 { window.draw(body); };
```

5.14.3.3 getCollider()

```
Collider Wall::getCollider ( )  [inline]
```

Returns collider

Definition at line 23 of file wall.hpp.

```
00023 { return Collider(body); }
```

5.14.3.4 Move2()

```
void Wall::Move2 (
            sf::Vector2f dir )
```

Moves the wall

Definition at line 15 of file wall.cpp.

```
00015 { body.move(dir); }
```

5.14.4 Member Data Documentation

5.14.4.1 body

```
sf::RectangleShape Wall::body
```

Definition at line 25 of file wall.hpp.

5.14.4.2 type_

```
int Wall::type_
```

Definition at line 26 of file wall.hpp.

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

- src/wall.hpp
- src/wall.cpp

5.15 Weapon Class Reference

```
#include <weapon.hpp>
```

Public Member Functions

- Weapon (sf::Texture *texture, sf::Texture *proj_texture)
- ~Weapon ()
- Projectile Fire (sf::Vector2f fire_position, sf::Vector2f fire_trajectory)

Public Attributes

- sf::RectangleShape weapon_body_
- std::vector< Projectile > weapon_projectile_
- float fire_rate_ = 0.1

5.15.1 Detailed Description

Class for weapon

Definition at line 11 of file weapon.hpp.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Weapon()

```
Weapon::Weapon (
            sf::Texture * texture,
            sf::Texture * proj_texture )
```

Constructor

Definition at line 7 of file weapon.cpp.

```
00007                                                                   {
00008   weapon_body_.setTexture(texture);
00009   Projectile new_projectile = Projectile(proj_texture);
00010   weapon_projectile_.push_back(new_projectile);
00011 };
```

5.15.2.2 ~Weapon()

```
Weapon::~Weapon ( )
```

Default destructor

Definition at line 13 of file weapon.cpp.

```
00013 {};
```

5.15.3 Member Function Documentation

5.15.3.1 Fire()

```
Projectile Weapon::Fire (
            sf::Vector2f fire_position,
            sf::Vector2f fire_trajectory )
```

Creates projectile

Definition at line 24 of file weapon.cpp.

```
00024                                                                                                   {
00025   Projectile proj(weapon_projectile_[0].projectile_texture_);
00026   proj.active_ = true;
00027   proj.body_.setTexture(proj.projectile_texture_);
00028   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00029   proj.body_.setPosition(fire_position);
00030   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00031   proj.projectile_trajectory_ = fire_trajectory/* * proj.velocity_multiplier_*/;
00032   proj.body_.move(proj.projectile_trajectory_);
00033   return proj;
00034 };
```

5.15.4 Member Data Documentation

5.15.4.1 fire_rate_

```
float Weapon::fire_rate_ = 0.1
```

Definition at line 24 of file weapon.hpp.

5.15.4.2 weapon_body_

```
sf::RectangleShape Weapon::weapon_body_
```

Definition at line 21 of file weapon.hpp.

5.15.4.3 weapon_projectile_

```
std::vector<Projectile> Weapon::weapon_projectile_
```

Definition at line 23 of file weapon.hpp.

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

- src/weapon.hpp
- src/weapon.cpp

Chapter 6

File Documentation

6.1 src/animation.cpp File Reference

`#include "animation.hpp"`
Include dependency graph for animation.cpp:

6.2 animation.cpp

```
00001 #include "animation.hpp"
00002
00003 /*
00004 The constructor of the animation class.
00005
00006 Parameters:
00007     sprite_sheets_path: the path for the sprite sheeth that contains the frames for the animation,
00008     for example: "libs/images/pumpkin_dude.png"
00009
00010     texture_crop_scale_x_: Integer describing how many pixels in the x direction is one frame in the
      sprite sheet cropped.
00011
00012     texture_crop_scale_y_: Integer describing how many pixels in the y direction is one frame in the
      sprite sheet cropped.
00013
00014 How the constructor works:
00015     This constructor creates the frames using the sprite sheets, and then pushes the frames to vector
      animation_frames_.
00016     This class uses animation_frames_ to animate an object frame by frame, or to "step" one frame
      forward repeatedly.
00017 */
00018 Animation::Animation(const std::string sprite_sheets_path, int texture_crop_scale_x_, int
      texture_crop_scale_y_) {
00019   sf::Texture temp_texture;
00020   temp_texture.loadFromFile(sprite_sheets_path);
00021   sf::Vector2u texture_size = temp_texture.getSize();
00022   int amount_of_x_offsets = (int(texture_size.x) / texture_crop_scale_x_);
00023   int amount_of_y_offsets = (int(texture_size.y) / texture_crop_scale_y_);
00024   int x_offset = 0; int y_offset = 0;
00025   for (int j = 0; j < amount_of_y_offsets; j++) {
00026     y_offset = j*texture_crop_scale_y_;
00027     for (int i = 0; i < amount_of_x_offsets; i++) {
00028       x_offset = i*texture_crop_scale_x_;
00029       sf::IntRect cropping(x_offset, y_offset, texture_crop_scale_x_, texture_crop_scale_y_);
00030       sf::Texture frame_texture;
00031       frame_texture.loadFromFile(sprite_sheets_path, cropping);
00032       animation_frames_.push_back(frame_texture);
00033     }
00034   }
00035   last_frame_ = animation_frames_.size() - 1;
00036 };
00037
00038 /* Destructor. */
00039 Animation::~Animation() {};
00040
00041 /* Sets the animation speed. */
```

```
00042 void Animation::setAnimationSpeed(float new_speed) {animation_speed_ = new_speed;};
00043
00044 /*
00045 This function is called to get the current frame that the animation is meant to display.
00046 Call example:
00047 body_.setTexture(animation01.getCurrentFrame());
00048 */
00049 sf::Texture* Animation::getCurrentFrame() {return &animation_frames_[current_frame_];};
00050
00051 /*
00052 Simply resets the animation to start from the beggining,
00053 useful for ex. when character stops walking to stand still instead.
00054 */
00055 void Animation::resetCurrentFrame() {current_frame_ = 0;};
00056
00057 /*
00058 Updates the animation with respect to SFML window time in seconds.
00059 Uses time to determine wether or not the next frame in the animation is going to be displayed.
00060 Returns true if frame was updated, returns false otherwise.
00061 */
00062 bool Animation::Update(float time) {
00063   actionTimer_ += time;
00064   if(actionTimer_ >= animation_speed_){
00065     actionTimer_ = 0;
00066     current_frame_ += 1;
00067     if(current_frame_ > last_frame_){
00068       current_frame_ = 0;
00069     }
00070     return true;
00071   }
00072   return false;
00073 };
```

6.3 src/animation.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <string>
```

Include dependency graph for animation.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Animation

6.4 animation.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <string>
00006
00007 #ifndef ANIMATION_CLASS
00008 #define ANIMATION_CLASS
00009
00010 class Animation {
00011  public:
00012   /**Animation constructor*/
00013   Animation(const std::string sprite_sheets_path, int frame_x_pos, int frame_y_pos);
00014   /**Default destructor*/
00015   ~Animation();
00016   /**Sets the animation speed to new_speed value*/
00017   void setAnimationSpeed(float new_speed);
00018   /**Returns current frame of the animation image as texture*/
00019   sf::Texture* getCurrentFrame();
00020   /**Resets the current frame*/
00021   void resetCurrentFrame();
00022   /**Updates the frame*/
00023   bool Update(float time);
```

```
00024
00025   std::vector<sf::Texture> animation_frames_;
00026   unsigned int current_frame_ = 0;
00027   unsigned int last_frame_;
00028   float animation_speed_ = 0.2; //Animation speed in frames per second
00029   float actionTimer_;
00030 };
00031
00032 #endif
```

6.5 src/collider.cpp File Reference

`#include "collider.hpp"`
Include dependency graph for collider.cpp:

6.6 collider.cpp

```
00001 #include "collider.hpp"
00002
00003 Collider::Collider(sf::RectangleShape& body) : body_(body) {}
00004
00005 Collider::~Collider() {}
00006
00007 // checks of collider intersects witch collider given as parameter
00008 // moves one if necessary
00009 bool Collider::checkCollider(Collider& other, float push) {
00010   // get positions
00011   sf::Vector2f otherPosition = other.getPosition();
00012   sf::Vector2f otherHalfSize = other.getHalfSize();
00013   sf::Vector2f thisPosition = getPosition();
00014   sf::Vector2f thisHalfSize = getHalfSize();
00015
00016   // find differences in location
00017   float xdif = otherPosition.x - thisPosition.x;
00018   float ydif = otherPosition.y - thisPosition.y;
00019
00020   float intersectX = abs(xdif) - (otherHalfSize.x + thisHalfSize.x);
00021   float intersectY = abs(ydif) - (otherHalfSize.y + thisHalfSize.y);
00022
00023   // if colliders intersect, move one of them based on push parameter and return
00024   // true
00025   if (intersectX < 0.0f && intersectY < 0.0f) {
00026     push = std::min(std::max(push, 0.0f), 1.0f);
00027
00028     if (abs(intersectX) < abs(intersectY)) {
00029       if (xdif > 0.0f) {
00030         Move(intersectX * (1.0f - push), 0.0f);
00031         other.Move(-intersectX * push, 0.0f);
00032       } else {
00033         Move(-intersectX * (1.0f - push), 0.0f);
00034         other.Move(intersectX * push, 0.0f);
00035       }
00036     } else {
00037       if (ydif > 0.0f) {
00038         Move(0.0f, intersectY * (1.0f - push));
00039         other.Move(0.0f, -intersectY * push);
00040       } else {
00041         Move(0.0f, -intersectY * (1.0f - push));
00042         other.Move(0.0f, intersectY * push);
00043       }
00044     }
00045
00046     return true;
00047   }
00048
00049   return false;
00050 }
```

6.7 src/collider.hpp File Reference

`#include "SFML/Graphics.hpp"`
Include dependency graph for collider.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Collider

6.8 collider.hpp

```
00001 #ifndef COLLIDER_CLASS
00002 #define COLLIDER_CLASS
00003
00004 #include "SFML/Graphics.hpp"
00005 /**Class for handling collisions*/
00006 class Collider {
00007  public:
00008   /**Constructor*/
00009   Collider(sf::RectangleShape& body);
00010   /**Default destructor*/
00011   ~Collider();
00012   /**Moves object out of collision */
00013   void Move(float dx, float dy) { body_.move(dx, dy); }
00014   /**Checks if two colliders are colliding*/
00015   bool checkCollider(Collider& other, float push);
00016   /**Returns collider position*/
00017   sf::Vector2f getPosition() { return body_.getPosition(); }
00018   /**Returns collider size divided by 2*/
00019   sf::Vector2f getHalfSize() { return body_.getSize() / 2.0f; }
00020
00021  private:
00022   sf::RectangleShape& body_;
00023 };
00024
00025 #endif
```

6.9 src/enemy.cpp File Reference

```
#include "enemy.hpp"
#include "projectile.hpp"
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <iostream>
```

Include dependency graph for enemy.cpp:

6.10 enemy.cpp

```
00001 #include "enemy.hpp"
00002 #include "projectile.hpp"
00003
00004 #include <stdlib.h> /* srand, rand, RAND_MAX */
00005 #include <time.h>   /* time */
00006 #include <math.h>   /* sqrt, pow */
00007
00008 #include <iostream>
00009
00010 Enemy::Enemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed)
00011     : speed_(speed) {
00012   body_.setSize(sf::Vector2f(40.0f, 40.0f));
00013   body_.setOrigin(body_.getSize() / 2.0f);
00014   body_.setPosition(spawnPos);
00015   enemy_animations_.push_back(*enemy_animation);
00016   body_.setTexture(&(enemy_animation->animation_frames_[0]));
00017 };
00018
00019 Enemy::~Enemy() {};
00020
00021 sf::Vector2f Enemy::getPosition() { return body_.getPosition(); };
00022
00023 bool Enemy::isRanged() { return is_ranged_type_; };
00024
00025 void Enemy::Draw(sf::RenderWindow& window) { window.draw(body_); };
```

```
00026
00027 void Enemy::Damage(float damage) {
00028   hp_ -= damage;
00029   if (hp_ <= 0) {alive_ = false;} //std::cout « "The NPC has died!" « std::endl;
00030 }
00031
00032 void Enemy::Update2(float time, sf::Vector2f player_position, std::vector<Projectile>&
       active_projectiles) {};
00033
00034 void Enemy::MoveRoom(int direction) {
00035   sf::Vector2f dir;
00036   if (direction == 0) {
00037     dir = sf::Vector2f(0, -10 * 50);
00038   }
00039   if (direction == 1) {
00040     dir = sf::Vector2f(10 * 50, 0);
00041   }
00042   if (direction == 2) {
00043     dir = sf::Vector2f(0, 10 * 50);
00044   }
00045   if (direction == 3) {
00046     dir = sf::Vector2f(-10 * 50, 0);
00047   }
00048   body_.move(dir);
00049 }
00050
00051
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00052
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00053
00054 ChasingEnemy::ChasingEnemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed)
00055  : Enemy(enemy_animation, spawnPos, speed) {};
00056
00057 ChasingEnemy::~ChasingEnemy() {};
00058
00059 void ChasingEnemy::Update(float time, sf::Vector2f player_position) {
00060   currentSpeed_ = speed_;
00061   actionTimer_ += time;
00062   //std::cout « "Interval timer: " « actionTimer_ « "\n";
00063
00064   /* Change walk direction to some random direction at random intervals */
00065
00066   if(actionTimer_ >= walkInterval_){
00067     actionTimer_ = 0.0f;
00068     walk_ = walk_ ? false : true; //tenary operator
00069
00070     /*
00071     walkInterval_ is a float between 0.5 and 2.5,
00072     this is how it is generated:
00073
00074     rand() gives a number between 0 and RAND_MAX,
00075     so divide by RAND_MAX to get a float between 0 and 1,
00076     or in this case 0 and 2, because it is multiplied by 2,
00077     and then plus 0.5 of that to get float between 0.5 and 2.5.
00078     */
00079     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00080
00081     if(walk_ == false){
00082       enemy_animations_[0].resetCurrentFrame();
00083       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00084     } else{
00085       attacking_player_ = (rand() % 3) == 1; //Enemies randomly attack sometimes
00086       walkInterval_ = 0.7;
00087     }
00088
00089     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00090     if(attacking_player_ == true) {
00091       sf::Vector2f direction_vector = player_position - body_.getPosition();
00092       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00093       direction_vector.x = direction_vector.x / vector_magnitude;
00094       direction_vector.y = direction_vector.y / vector_magnitude;
00095       //std::cout « direction_vector.x « " " « direction_vector.y « " " « vector_magnitude « "\n";
00096       movement.x *= time * direction_vector.x * 2.f;
00097       movement.y *= time * direction_vector.y * 2.f;
00098     } else {
00099       float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00100       float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00101       movement.x *= random_float_x * time;
00102       movement.y *= random_float_y * time;
00103     }
00104     direction_ = movement;
00105   }
00106   if(walk_){
00107     if(enemy_animations_[0].Update(time)){
00108       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00109     }
```

```
00110     body_.move(direction_);
00111     /*Collider col = this->getCollider();
00112     for (auto room : rooms) {
00113       for (auto wall : room.walls) {
00114         for (auto w : wall) {
00115           if (w.type_ == 1) {
00116             Collider wallCol = w.getCollider();
00117             wallCol.checkCollider(col, 1);
00118           }
00119         }
00120       }
00121     }*/
00122   }
00123 };
00124
00125
      ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00126
      ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00127
00128 RangedEnemy::RangedEnemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed,
00129 sf::Texture* enemy_projectile_texture)
00130  : Enemy(enemy_animation, spawnPos, speed), enemy_projectile_texture_(enemy_projectile_texture)/*,
      is_ranged_type_(true)*/ {
00131     is_ranged_type_ = true;
00132     //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00133  };
00134
00135 RangedEnemy::~RangedEnemy() {};
00136
00137 void RangedEnemy::Update(float time, sf::Vector2f player_position) {
00138   currentSpeed_ = speed_;
00139   actionTimer_ += time;
00140
00141   /* Change walk direction to some random direction at random intervals */
00142
00143   if(actionTimer_ >= walkInterval_){
00144     actionTimer_ = 0.0f;
00145     walk_ = walk_ ? false : true; //tenary operator
00146
00147     /*
00148     walkInterval_ is a float between 0.5 and 2.5,
00149     this is how it is generated:
00150
00151     rand() gives a number between 0 and RAND_MAX,
00152     so divide by RAND_MAX to get a float between 0 and 1,
00153     or in this case 0 and 2, because it is multiplied by 2,
00154     and then plus 0.5 of that to get float between 0.5 and 2.5.
00155     */
00156     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00157
00158     if(walk_ == false){
00159       enemy_animations_[0].resetCurrentFrame();
00160       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00161     }
00162
00163     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00164     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00165     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00166     movement.x *= random_float_x * time;
00167     movement.y *= random_float_y * time;
00168     direction_ = movement;
00169   }
00170
00171   if(walk_){
00172     if(enemy_animations_[0].Update(time)){
00173       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00174     }
00175     body_.move(direction_);
00176     /*Collider col = this->getCollider();
00177     for (auto room : rooms) {
00178       for (auto wall : room.walls) {
00179         for (auto w : wall) {
00180           if (w.type_ == 1) {
00181             Collider wallCol = w.getCollider();
00182             wallCol.checkCollider(col, 1);
00183           }
00184         }
00185       }
00186     }*/
00187   }
00188 };
00189
00190 void RangedEnemy::Update2(float time, sf::Vector2f player_position, std::vector<Projectile>&
      active_projectiles) {
00191   currentSpeed_ = speed_;
00192   actionTimer_ += time;
```

```
00193
00194   /* Change walk direction to some random direction at random intervals */
00195
00196   if(actionTimer_ >= walkInterval_){
00197     actionTimer_ = 0.0f;
00198     walk_ = walk_ ? false : true; //tenary operator
00199
00200     /*
00201     walkInterval_ is a float between 0.5 and 2.5,
00202     this is how it is generated:
00203
00204     rand() gives a number between 0 and RAND_MAX,
00205     so divide by RAND_MAX to get a float between 0 and 1,
00206     or in this case 0 and 2, because it is multiplied by 2,
00207     and then plus 0.5 of that to get float between 0.5 and 2.5.
00208     */
00209     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00210
00211     if(walk_ == false){
00212       enemy_animations_[0].resetCurrentFrame();
00213       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00214     }
00215
00216     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00217     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00218     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00219     movement.x *= random_float_x * time;
00220     movement.y *= random_float_y * time;
00221     direction_ = movement;
00222
00223     if(rand() % 2) { //Shoot somewhere: 50% chance
00224       sf::Vector2f shoot_direction(10, 10);
00225       if(rand() % 4 == 1) { //Shoot at random direction: 25% chance
00226         random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00227         random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00228         shoot_direction.x *= random_float_x;
00229         shoot_direction.y *= random_float_y;
00230       } else { //Shoot at direction of player: 75% chance
00231         sf::Vector2f direction_vector = player_position - body_.getPosition();
00232         float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00233         shoot_direction.x *= direction_vector.x / vector_magnitude;
00234         shoot_direction.y *= direction_vector.y / vector_magnitude;
00235       }
00236       Projectile new_projectile = this->ShootAtDirection(shoot_direction);
00237       active_projectiles.push_back(new_projectile);
00238     }
00239   }
00240   if(walk_){
00241     if(enemy_animations_[0].Update(time)){
00242       body_.setTexture(enemy_animations_[0].getCurrentFrame());
00243     }
00244     body_.move(direction_);
00245     /*Collider col = this->getCollider();
00246     for (auto room : rooms) {
00247       for (auto wall : room.walls) {
00248         for (auto w : wall) {
00249           if (w.type_ == 1) {
00250             Collider wallCol = w.getCollider();
00251             wallCol.checkCollider(col, 1);
00252           }
00253         }
00254       }
00255     }*/
00256   }
00257 };
00258
00259 Projectile RangedEnemy::ShootAtDirection(sf::Vector2f direction) {
00260   //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00261   //Projectile Weapon::Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory)
00262   Projectile proj(this->enemy_projectile_texture_);
00263   //proj.friendly_[0] = 0;
00264   proj.active_ = true;
00265   proj.body_.setTexture(proj.projectile_texture_);
00266   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00267   proj.body_.setPosition(body_.getPosition());
00268   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00269   proj.projectile_trajectory_ = direction/* * proj.velocity_multiplier_*/;
00270   proj.body_.move(proj.projectile_trajectory_);
00271   return proj;
00272 };
00273
00274
      ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00275
      ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00276
00277 FinalBoss::FinalBoss(Animation *enemy_animation, sf::Vector2f spawnPos,
```

```
00278                     float speed, sf::Texture* enemy_projectile_texture)
00279     : Enemy(enemy_animation, spawnPos, speed), enemy_projectile_texture_(enemy_projectile_texture) {
00280 is_ranged_type_ = true;
00281 walk_ = true;
00282 body_.setSize(sf::Vector2f(100.0f, 100.0f));
00283 body_.setOrigin(body_.getSize() / 2.0f);
00284 };
00285
00286 void FinalBoss::Update(float time, sf::Vector2f player_position) {
00287   currentSpeed_ = speed_;
00288   actionTimer_ += time;
00289   //std::cout << "Interval timer: " << actionTimer_ << "\n";
00290
00291   /* Change walk direction to some random direction at random intervals */
00292
00293   if(actionTimer_ >= walkInterval_){
00294     actionTimer_ = 0.0f;
00295
00296     /*
00297     walkInterval_ is a float between 0.5 and 2.5,
00298     this is how it is generated:
00299
00300     rand() gives a number between 0 and RAND_MAX,
00301     so divide by RAND_MAX to get a float between 0 and 1,
00302     or in this case 0 and 2, because it is multiplied by 2,
00303     and then plus 0.5 of that to get float between 0.5 and 2.5.
00304     */
00305     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00306
00307     attacking_player_ = (rand() % 3) == 1; //Enemies randomly attack sometimes
00308     walkInterval_ = 0.7;
00309
00310     sf::Vector2f movement(currentSpeed_, currentSpeed_);
00311     if(attacking_player_ == true) {
00312       sf::Vector2f direction_vector = player_position - body_.getPosition();
00313       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00314       direction_vector.x = direction_vector.x / vector_magnitude;
00315       direction_vector.y = direction_vector.y / vector_magnitude;
00316       //std::cout << direction_vector.x << " " << direction_vector.y << " " << vector_magnitude << "\n";
00317       movement.x *= time * direction_vector.x * 2.f;
00318       movement.y *= time * direction_vector.y * 2.f;
00319     } else {
00320       float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00321       float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00322       movement.x *= random_float_x * time;
00323       movement.y *= random_float_y * time;
00324     }
00325     direction_ = movement;
00326   }
00327   if(enemy_animations_[0].Update(time)){
00328     body_.setTexture(enemy_animations_[0].getCurrentFrame());
00329   }
00330   body_.move(direction_);
00331     /*Collider col = this->getCollider();
00332     for (auto room : rooms) {
00333       for (auto wall : room.walls) {
00334         for (auto w : wall) {
00335           if (w.type_ == 1) {
00336             Collider wallCol = w.getCollider();
00337             wallCol.checkCollider(col, 1);
00338           }
00339         }
00340       }
00341     }*/
00342 };
00343
00344 void FinalBoss::Update2(float time, sf::Vector2f player_position, std::vector<Projectile>&
       active_projectiles) {
00345   currentSpeed_ = speed_;
00346   actionTimer_ += time;
00347
00348   /* Change walk direction to some random direction at random intervals */
00349
00350   if(actionTimer_ >= walkInterval_) {
00351     actionTimer_ = 0.0f;
00352     /*
00353     walkInterval_ is a float between 0.5 and 2.5,
00354     this is how it is generated:
00355
00356     rand() gives a number between 0 and RAND_MAX,
00357     so divide by RAND_MAX to get a float between 0 and 1,
00358     or in this case 0 and 2, because it is multiplied by 2,
00359     and then plus 0.5 of that to get float between 0.5 and 2.5.
00360     */
00361     walkInterval_ = ((float(rand()) / float(RAND_MAX)) * 2) + 0.5f;
00362
00363     sf::Vector2f movement(currentSpeed_, currentSpeed_);
```

```
00364     float random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00365     float random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00366     movement.x *= random_float_x * time;
00367     movement.y *= random_float_y * time;
00368     direction_ = movement;
00369
00370     if(rand() % 2) {
00371       sf::Vector2f shoot_direction1(10, 10);
00372       random_float_x = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00373       random_float_y = ((float(rand()) / float(RAND_MAX)) * 2) - 1.0f;
00374       shoot_direction1.x *= random_float_x;
00375       shoot_direction1.y *= random_float_y;
00376       Projectile new_projectile1 = this->ShootAtDirection(shoot_direction1);
00377       active_projectiles.push_back(new_projectile1);
00378     }
00379     if(rand() % 2) {
00380       sf::Vector2f shoot_direction2(7, 7);
00381       sf::Vector2f direction_vector = player_position - body_.getPosition();
00382       float vector_magnitude = sqrt(pow(direction_vector.x, 2.0) + pow(direction_vector.y, 2.0));
00383       shoot_direction2.x *= direction_vector.x / vector_magnitude;
00384       shoot_direction2.y *= direction_vector.y / vector_magnitude;
00385       Projectile new_projectile2 = this->ShootAtDirection(shoot_direction2);
00386       active_projectiles.push_back(new_projectile2);
00387     }
00388   }
00389   if(enemy_animations_[0].Update(time)) {
00390     body_.setTexture(enemy_animations_[0].getCurrentFrame());
00391   }
00392   body_.move(direction_);
00393     /*Collider col = this->getCollider();
00394     for (auto room : rooms) {
00395       for (auto wall : room.walls) {
00396         for (auto w : wall) {
00397           if (w.type_ == 1) {
00398             Collider wallCol = w.getCollider();
00399             wallCol.checkCollider(col, 1);
00400           }
00401         }
00402       }
00403     }*/
00404 };
00405
00406 Projectile FinalBoss::ShootAtDirection(sf::Vector2f direction) {
00407   //enemy_projectile_ = std::make_unique<Projectile>(enemy_projectile_texture);
00408   //Projectile Weapon::Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory)
00409   Projectile proj(this->enemy_projectile_texture_);
00410   //proj.friendly_[0] = 0;
00411   proj.active_ = true;
00412   proj.body_.setTexture(proj.projectile_texture_);
00413   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00414   proj.body_.setPosition(body_.getPosition());
00415   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00416   proj.projectile_trajectory_ = direction/* * proj.velocity_multiplier_*/;
00417   proj.body_.move(proj.projectile_trajectory_);
00418   return proj;
00419 };
```

6.11 src/enemy.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <vector>
#include "animation.hpp"
#include "collider.hpp"
```

Include dependency graph for enemy.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Enemy
- class ChasingEnemy
- class RangedEnemy
- class FinalBoss

6.12 enemy.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <vector>
00006 //#include <memory>
00007
00008 #include "animation.hpp"
00009 #include "collider.hpp"
00010
00011 #ifndef ENEMY_CLASS
00012 #define ENEMY_CLASS
00013
00014 class Projectile; //Forward declaration is used here to avoid circular dependency with projectile
       class
00015
00016 /**Virtual class for enemies*/
00017 class Enemy {
00018  public:
00019   /**Constructor*/
00020   Enemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed);
00021   /**Default destructor*/
00022   ~Enemy();
00023   /**Returns enemy position*/
00024   sf::Vector2f getPosition();
00025   /**Check if enemy is ranged*/
00026   bool isRanged();
00027   /**Returns enemy collider*/
00028   Collider getCollider()
00029   {
00030     return Collider(body_);
00031   };
00032   /**Moves enemy when player is traversing between rooms*/
00033   void MoveRoom(int dir);
00034   /**Used for rendering the enemy*/
00035   void Draw(sf::RenderWindow& window);
00036   /**Reduces enemy hp*/
00037   void Damage(float dmg);
00038   /**Virtual function for updating enemy position*/
00039   virtual void Update(float time, sf::Vector2f player_position) = 0;
00040
00041   virtual void Update2(float time, sf::Vector2f player_position, std::vector<Projectile>&
       active_projectiles);
00042
00043   bool walk_ = false;
00044   bool attacking_player_ = false;
00045   bool is_ranged_type_ = false;
00046   float speed_ = 100;
00047   float hp_ = 100;
00048   float damage_ = 1;
00049   float currentSpeed_;
00050   float actionTimer_;
00051   float walkInterval_ = 2;
00052   bool alive_ = true;
00053   sf::RectangleShape body_;
00054   sf::Vector2f direction_;
00055   std::vector<Animation> enemy_animations_;
00056 };
00057
00058
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00059
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00060 /**Class for enemy that chases the player*/
00061 class ChasingEnemy : public Enemy{
00062  public:
00063   ChasingEnemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed);
00064
00065   ~ChasingEnemy();
00066
00067   virtual void Update(float time, sf::Vector2f player_position) override;
00068 };
00069
00070
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00071
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00072
00073 class Projectile;
00074 /**Class for enemy that shoots projectiles*/
00075 class RangedEnemy : public Enemy{
00076  public:
00077   RangedEnemy(Animation *enemy_animation, sf::Vector2f spawnPos, float speed,
00078   sf::Texture* enemy_projectile_texture);
00079
```

```
00080   ~RangedEnemy();
00081
00082   virtual void Update(float time, sf::Vector2f player_position) override;
00083
00084   void Update2(float time, sf::Vector2f player_position, std::vector<Projectile>& active_projectiles);
00085   /**Used to shoot projetiles towards direction*/
00086   Projectile ShootAtDirection(sf::Vector2f direction);
00087
00088   sf::Texture* enemy_projectile_texture_;
00089 };
00090
00091
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00092
       ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00093 /**Class for the finall Boss*/
00094 class FinalBoss : public Enemy{
00095 public:
00096
00097   FinalBoss(Animation *enemy_animation, sf::Vector2f spawnPos, float speed,
00098   sf::Texture* enemy_projectile_texture);
00099
00100   virtual void Update(float time, sf::Vector2f player_position) override;
00101
00102   void Update2(float time, sf::Vector2f player_position, std::vector<Projectile>& active_projectiles);
00103   /**Used to shoot projetiles towards direction*/
00104   Projectile ShootAtDirection(sf::Vector2f direction);
00105
00106   sf::Texture* enemy_projectile_texture_;
00107 };
00108 #endif
```

6.13 src/HUD.cpp File Reference

```
#include "HUD.hpp"
#include <sstream>
#include <string>
```

Include dependency graph for HUD.cpp:

6.14 HUD.cpp

```
00001 #include "HUD.hpp"
00002
00003 #include <sstream>
00004 #include <string>
00005
00006 HUD::HUD(sf::Font font, std::vector<sf::Texture*> hp_textures,
00007          sf::Texture* background, sf::Texture* inventory_texture,
00008          std::vector<sf::Texture*> item_textures)
00009     : font_(font), hp_textures_(hp_textures) {
00010   // Window size 500x500 assumed
00011   background_.setSize(sf::Vector2f(500.0f, 150.0f));
00012   background_.setPosition(sf::Vector2f(0.0f, 500.0f));
00013   background_.setTexture(background);
00014   for (unsigned int i = 0; i < 3; i++) {
00015     sf::RectangleShape heart;
00016     heart.setSize(sf::Vector2f(50.0f, 50.0f));
00017     heart.setPosition(sf::Vector2f((i + 1) * 50.0f, 520.0f));
00018     heart.setTexture(hp_textures_[0]);
00019     hp_bar_.push_back(heart);
00020   }
00021   for (unsigned int i = 0; i < 3; i++) {
00022     sf::Text text;
00023     text.setString("0");
00024     text.setCharacterSize(24);
00025     text.setPosition(sf::Vector2f((i + 5) * 50.0f, 570.0f));
00026     texts_.push_back(text);
00027   }
00028   for (unsigned int i = 4; i < 7; i++) {
00029     sf::RectangleShape inv_slot;
00030     inv_slot.setSize(sf::Vector2f(50.0f, 50.0f));
00031     inv_slot.setPosition(sf::Vector2f((i + 1) * 50.0f, 520.0f));
00032     inv_slot.setTexture(inventory_texture);
00033     inventory_bar_.push_back(inv_slot);
00034   }
00035   for (unsigned int i = 0; i < item_textures.size(); i++) {
```

```
00036     sf::RectangleShape item_image;
00037     item_image.setSize(sf::Vector2f(50.0f, 50.0f));
00038     item_image.setPosition(sf::Vector2f((i + 5) * 50.0f, 520.0f));
00039     item_image.setTexture(item_textures[i]);
00040     hud_items_.push_back(item_image);
00041   }
00042 }
00043
00044 void HUD::Display(sf::RenderWindow& window) {
00045   window.draw(background_);
00046   for (unsigned int i = 0; i < hp_bar_.size(); i++) {
00047     window.draw(hp_bar_[i]);
00048   }
00049   for (unsigned int j = 0; j < inventory_bar_.size(); j++) {
00050     window.draw(inventory_bar_[j]);
00051     texts_[j].setFont(font_);
00052     window.draw(texts_[j]);
00053   }
00054   if (hud_items_.size() != 0) {
00055     for (unsigned int i = 0; i < hud_items_.size(); i++) {
00056       window.draw(hud_items_[i]);
00057     }
00058   }
00059 }
00060
00061 void HUD::Update(std::vector<Item>& items, float playerHP) {
00062   int hp_pots = 0;
00063   int spd_pots = 0;
00064   int coins = 0;
00065   for (auto i : items) {
00066     if (i.type == "hp_pot") {
00067       hp_pots++;
00068     }
00069     if (i.type == "speed_pot") {
00070       spd_pots++;
00071     }
00072     if (i.type == "coin") {
00073       coins++;
00074     }
00075   }
00076   std::stringstream ss;
00077   ss « hp_pots;
00078   texts_[0].setString(ss.str());
00079   std::stringstream ss2;
00080   ss2 « spd_pots;
00081   texts_[1].setString(ss2.str());
00082   std::stringstream ss3;
00083   ss3 « coins;
00084   texts_[2].setString(ss3.str());
00085
00086   int hp = static_cast<int>(playerHP);
00087   switch (hp) {
00088     case 6:
00089       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[0]);
00090       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00091       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00092       break;
00093     case 5:
00094       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[1]);
00095       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00096       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00097       break;
00098     case 4:
00099       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00100       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[0]);
00101       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00102       break;
00103     case 3:
00104       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00105       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[1]);
00106       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00107       break;
00108     case 2:
00109       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00110       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00111       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[0]);
00112       break;
00113     case 1:
00114       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00115       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00116       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[1]);
00117       break;
00118     case 0:
00119       hp_bar_[hp_bar_.size() - 1].setTexture(hp_textures_[2]);
00120       hp_bar_[hp_bar_.size() - 2].setTexture(hp_textures_[2]);
00121       hp_bar_[hp_bar_.size() - 3].setTexture(hp_textures_[2]);
00122       break;
```

```
00123   }
00124 }
00125
00126 HUD::~HUD() {}
```

6.15 src/HUD.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <vector>
#include "item.hpp"
```

Include dependency graph for HUD.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class HUD

6.16 HUD.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <vector>
00006
00007 #include "item.hpp"
00008
00009 #ifndef HUD_CLASS
00010 #define HUD_CLASS
00011
00012 class HUD {
00013  public:
00014   /**Constructs HUD*/
00015   HUD(sf::Font font, std::vector<sf::Texture*> hp, sf::Texture* background,
00016       sf::Texture* inventory, std::vector<sf::Texture*> item_textures);
00017   /**Default destructor*/
00018   ~HUD();
00019   /**Rendrers the HUD*/
00020   void Display(sf::RenderWindow& window);
00021   /**Updates the HUD*/
00022   void Update(std::vector<Item>&, float);
00023   sf::RectangleShape background_;
00024   std::vector<sf::RectangleShape> hp_bar_;
00025   std::vector<sf::RectangleShape> inventory_bar_;
00026   std::vector<sf::RectangleShape> hud_items_;
00027   std::vector<sf::Text> texts_;
00028   sf::Font font_;
00029   std::vector<sf::Texture*> hp_textures_;
00030 };
00031 #endif
```

6.17 src/item.cpp File Reference

```
#include "item.hpp"
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for item.cpp:

6.18 item.cpp

```
00001 #include "item.hpp"
00002
00003 #include <SFML/System/Vector2.hpp>
00004 Item::Item(sf::Texture *texture, sf::Vector2f position, std::string type)
00005     : type(type) {
00006   body.setSize(sf::Vector2f(30, 30));
00007   body.setPosition(position);
00008   body.setTexture(texture);
00009   body.setOrigin(sf::Vector2f(30, 30) / 2.0f);
00010 };
00011 void Item::Draw(sf::RenderWindow &window) { window.draw(body); }
```

6.19 src/item.hpp File Reference

```
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <string>
#include "collider.hpp"
```

Include dependency graph for item.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Item

6.20 item.hpp

```
00001 #include <SFML/Graphics/RectangleShape.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <string>
00004
00005 #include "collider.hpp"
00006 #ifndef ITEM_CLASS
00007 #define ITEM_CLASS
00008
00009 /**Class for handling items*/
00010 class Item {
00011  public:
00012   /**Constructor for Item oject*/
00013   Item(sf::Texture* texture, sf::Vector2f position, std::string type);
00014   /**Default destructor for item*/
00015   ~Item(){};
00016   /**Used for rendering the items*/
00017   void Draw(sf::RenderWindow& window);
00018   /**Returns the item collider*/
00019   Collider getCollider() { return Collider(body); }
00020   sf::RectangleShape body;
00021   std::string type;
00022 };
00023 #endif
```

6.21 src/main.cpp File Reference

```
#include <stdlib.h>
#include <time.h>
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
```

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include "HUD.hpp"
#include "animation.hpp"
#include "collider.hpp"
#include "enemy.hpp"
#include "menu.hpp"
#include "player.hpp"
#include "projectile.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main ()

Variables

- const int WIDTH = 500
- const int HEIGHT = 650

6.21.1 Function Documentation

6.21.1.1 main()

```
int main ( )
```

Definition at line 45 of file main.cpp.

```
00045           {
00046   RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "SFML window");
00047   window.setFramerateLimit(30);  // Setting the framerate limit lower helps the
00048                                  // program be less CPU intensive.
00049   srand(time(NULL));
00050   // Font for text
00051   sf::Font font;
00052   if (!font.loadFromFile("libs/fonts/arial.ttf")) {
00053     std::cout « "Error loading font."
00054               « "\n";
00055   }
00056   sf::Text text;
00057   text.setFont(font);
00058   text.setString("GAME OVER");
00059   text.setCharacterSize(24);
00060   text.setPosition(WIDTH / 2, HEIGHT / 2);
00061
00062   // Music
00063   sf::Music music;
00064   if (!music.openFromFile("libs/audio/music.ogg")) {
00065     std::cout « "Error loading music."
00066               « "\n";
00067   }
00068   music.setRelativeToListener(true);
00069   music.setVolume(20);
00070   music.setLoop(true);
00071   music.play();
00072
00073   // Sounds
00074   sf::SoundBuffer punchbuf;
00075   punchbuf.loadFromFile("libs/audio/punch.ogg");
00076   sf::Sound punch;
00077   punch.setBuffer(punchbuf);
```

```
00078
00079   sf::SoundBuffer deathbuf;
00080   deathbuf.loadFromFile("libs/audio/death.ogg");
00081   sf::Sound death;
00082   death.setBuffer(deathbuf);
00083
00084   sf::SoundBuffer shootbuf;
00085   shootbuf.loadFromFile("libs/audio/shot.ogg");
00086   sf::Sound shoot;
00087   shoot.setBuffer(shootbuf);
00088
00089   sf::SoundBuffer drinkbuf;
00090   drinkbuf.loadFromFile("libs/audio/drink.ogg");
00091   sf::Sound drink;
00092   drink.setBuffer(drinkbuf);
00093
00094   sf::SoundBuffer pickbuf;
00095   pickbuf.loadFromFile("libs/audio/pickup.ogg");
00096   sf::Sound pickup;
00097   pickup.setBuffer(pickbuf);
00098
00099   // Textures
00100   sf::Texture* walltexture = new sf::Texture();
00101   walltexture->loadFromFile("libs/images/wall_left.png");
00102
00103   // sf::Texture enemytexture;
00104   // sf::IntRect enemytexture_crop(64, 11, 17, 21);
00105   // enemytexture.loadFromFile("libs/images/pumpkin_dude.png",
00106   // enemytexture_crop);
00107   Animation enemyAnimation("libs/images/pumpkin_dude.png", 16, 23);
00108   // sf::Texture enemytexture2;
00109   // sf::IntRect enemytexture_crop2(64, 11, 17, 21);
00110   // enemytexture2.loadFromFile("libs/images/skeleton_dude.png",
00111   // enemytexture_crop2);
00112   Animation enemyAnimation2("libs/images/skeleton_dude.png", 16, 20);
00113
00114   Animation boss_animation("libs/images/boss.png", 32, 36);
00115
00116   Texture weapon_texture;
00117   weapon_texture.loadFromFile("libs/images/simple gun.png");
00118   Texture projectile_texture;
00119   projectile_texture.loadFromFile("libs/images/simple bullet.png");
00120   Texture enemy_projectile_texture;
00121   enemy_projectile_texture.loadFromFile("libs/images/enemy bullet.png");
00122   sf::Texture playerTexture;
00123   playerTexture.loadFromFile("libs/images/character.png");
00124   sf::Texture* heart_texture = new sf::Texture();
00125   heart_texture->loadFromFile("libs/images/ui_heart_full.png");
00126   sf::Texture* heart_half_texture = new sf::Texture();
00127   heart_half_texture->loadFromFile("libs/images/ui_heart_half.png");
00128   sf::Texture* heart_empty_texture = new sf::Texture();
00129   heart_empty_texture->loadFromFile("libs/images/ui_heart_empty.png");
00130   sf::Texture* backgroundtexture = new sf::Texture();
00131   backgroundtexture->loadFromFile("libs/images/floor_1.png");
00132   sf::Texture player_death_texture;
00133   player_death_texture.loadFromFile("libs/images/character_grave.png");
00134
00135   sf::Texture healPotTexture;
00136   healPotTexture.loadFromFile("libs/images/flask_big_red.png");
00137   sf::Texture speedPotTexture;
00138   speedPotTexture.loadFromFile("libs/images/flask_big_blue.png");
00139   sf::Texture coinTexture;
00140   coinTexture.loadFromFile("libs/images/coin.png");
00141   sf::Texture* healPotTexture2 = new sf::Texture();
00142   healPotTexture2->loadFromFile("libs/images/flask_big_red.png");
00143   sf::Texture* speedPotTexture2 = new sf::Texture();
00144   speedPotTexture2->loadFromFile("libs/images/flask_big_blue.png");
00145   sf::Texture* coinTexture2 = new sf::Texture();
00146   coinTexture2->loadFromFile("libs/images/coin.png");
00147   std::vector<sf::Texture*> item_textures = {healPotTexture2, speedPotTexture2,
00148                                              coinTexture2};
00149   std::vector<sf::Texture*> heart_textures = {heart_texture, heart_half_texture,
00150                                               heart_empty_texture};
00151   Menu menu(font, walltexture);
00152
00153   // Generate map and player
00154   HUD hud(font, heart_textures, backgroundtexture, backgroundtexture,
00155           item_textures);
00156   // Map map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00157   // senemy_projectile_texture);
00158
00159   std::vector<Projectile> active_projectiles;  // Keep track of projectiles that
00160                                                // exist inside the SFML window.
00161
00162   // Generate map and player
00163   Map map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00164           &enemy_projectile_texture);
```

```
00165
00166   Weapon player_starting_weapon = Weapon(&weapon_texture, &projectile_texture);
00167
00168   float player_speed = 200;
00169   Vector2f spawnPos = Vector2f(100.0f, 100.0f);
00170   Player player(&playerTexture, &player_death_texture, spawnPos, player_speed,
00171                 &player_starting_weapon);
00172
00173   Clock clock;
00174   int resetCounter = 1;
00175   // This while loop runs as long as window is open. In this case it runs until
00176   // window.close(); is called.
00177   while (window.isOpen()) {
00178     Event event;
00179     if (menu.active_) {
00180       while (window.pollEvent(event)) {
00181         if (event.type == Event::Closed) {
00182           window.close();
00183         }  // This event closes the window when close button is pressed.
00184       }
00185       resetCounter = 0;
00186       window.clear();
00187       menu.Display(window);
00188       window.display();
00189     } else if (resetCounter == 0) {
00190       srand(clock.restart().asMilliseconds());
00191       map = Map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00192                 &enemy_projectile_texture);
00193       player = Player(&playerTexture, &player_death_texture, spawnPos,
00194                       player_speed, &player_starting_weapon);
00195       active_projectiles.clear();
00196       resetCounter++;
00197     } else {
00198       float time = clock.restart().asSeconds();
00199
00200       // This while loop goes through all window events, such as key presses
00201       // and mouse presses.
00202
00203       while (window.pollEvent(event)) {
00204         if (event.type == Event::Closed) {
00205           window.close();
00206         }  // This event closes the window when close button is pressed.
00207       }
00208
00209       std::vector<int> sounds = player.Update(time, map, active_projectiles);
00210       if (sounds[0]) {
00211         shoot.play();
00212       }
00213       if (sounds[1]) {
00214         punch.play();
00215       }
00216       if (sounds[2]) {
00217         drink.play();
00218       }
00219       if (sounds[3]) {
00220         pickup.play();
00221       }
00222       // update enemies and remove them if they are dead
00223       for (auto& enemy : map.enemies) {
00224         if (enemy->isRanged()) {
00225           enemy->Update2(time, player.getPosition(), active_projectiles);
00226         } else {
00227           enemy->Update(time, player.getPosition());
00228         }
00229         for (unsigned int i = 0; i < map.enemies.size(); i++) {
00230           if (!map.enemies[i]->alive_) {
00231             map.SpawnItem(i, &healPotTexture, &speedPotTexture, &coinTexture);
00232             map.enemies.erase(map.enemies.begin() + i);
00233             death.play();
00234           }
00235         }
00236       }
00237
00238       // update active projectiles and remove them if they have hit a wall or
00239       // an enemy
00240       for (auto& proj : active_projectiles) {
00241         // sf::Vector2u secret_size = proj.projectile_texture_->getSize();
00242         // std::cout << secret_size.x << " " << secret_size.y << "\n";
00243         if (proj.Update(time, map, player)) {
00244           proj.deActivate();
00245           for (unsigned int i = 0; i < active_projectiles.size(); i++) {
00246             if (!active_projectiles[i].active_) {
00247               active_projectiles.erase(active_projectiles.begin() + i);
00248             }
00249           }
00250         }
00251       }
```

```
00252       // std::cout « active_projectiles.size() « "\n";
00253
00254       window.clear();
00255       Collider playerCollider = player.getCollider();
00256       map.Display(window, playerCollider);
00257       player.Draw(window);
00258
00259       for (auto& enemy : map.enemies) {
00260         enemy->Draw(window);
00261       }
00262       for (auto& proj : active_projectiles) {
00263         proj.Draw(window);
00264       }
00265
00266       hud.Display(window);
00267       hud.Update(player.items_, player.hp_);
00268
00269       if (player.isAlive() == false) {
00270         // window.draw(text);
00271         // window.clear();
00272         menu.active_ = true;
00273       }
00274       window.display();
00275     }
00276   }
00277
00278   // for (auto& enemy : map.enemies) {
00279   //   delete enemy;
00280   // }
00281   delete walltexture;
00282   delete backgroundtexture;
00283   return 0;
00284 }
```

6.21.2 Variable Documentation

6.21.2.1 HEIGHT

```
const int HEIGHT = 650
```

Definition at line 24 of file main.cpp.

6.21.2.2 WIDTH

```
const int WIDTH = 500
```

Definition at line 23 of file main.cpp.

6.22 main.cpp

```
00001 // https://github.com/SFML/SFML/issues/1673
00002 #include <stdlib.h>
00003 #include <time.h>
00004
00005 #include <SFML/Audio.hpp>
00006 #include <SFML/Graphics.hpp>
00007 #include <SFML/Graphics/Texture.hpp>
00008 #include <SFML/System.hpp>
00009 #include <SFML/Window.hpp>
00010 #include <cstdlib>
00011 #include <iostream>
00012 #include <vector>
```

```
00013
00014 #include "HUD.hpp"
00015 #include "animation.hpp"
00016 #include "collider.hpp"
00017 #include "enemy.hpp"
00018 #include "menu.hpp"
00019 #include "player.hpp"
00020 #include "projectile.hpp"
00021 using namespace sf;
00022
00023 const int WIDTH = 500;
00024 const int HEIGHT = 650;
00025
00026 /*
00027 Here is a simple example setup for SFML code, which shows how SFML works, this
00028 main file will be modified as new classes and functionalities are introduced
00029 into the project. But until that this code can be used to test wether SFML works
00030 on the local machine or not.
00031
00032 This code can be run by calling the command "make" in the VSCode terminal after
00033 make is installed and then the local folder should create an executable .exe
00034 file, which can be run by double clicking it in the project file path.
00035
00036 In order to run SFML, SFML and mingw32 have to be downloaded from the SFML
00037 website. After this the downloaded folders have to be put in some file path that
00038 matches the one in the Makefile(SFML_DOWNLOAD_PATH and MINGW_DOWNLOAD_PATH). The
00039 bin folders inside mingw32 and SFML-2.5.1 have to be added into environment file
00040 variables: On your system search for Edit environment variables for your account
00041 --> from user variables select Path and edit --> click on new and copy the bin
00042 folder filepaths there
00043 */
00044 // TODO: negative y direction collision, player damage cooldown
00045 int main() {
00046   RenderWindow window(sf::VideoMode(WIDTH, HEIGHT), "SFML window");
00047   window.setFramerateLimit(30);  // Setting the framerate limit lower helps the
00048                                  // program be less CPU intensive.
00049   srand(time(NULL));
00050   // Font for text
00051   sf::Font font;
00052   if (!font.loadFromFile("libs/fonts/arial.ttf")) {
00053     std::cout « "Error loading font."
00054               « "\n";
00055   }
00056   sf::Text text;
00057   text.setFont(font);
00058   text.setString("GAME OVER");
00059   text.setCharacterSize(24);
00060   text.setPosition(WIDTH / 2, HEIGHT / 2);
00061
00062   // Music
00063   sf::Music music;
00064   if (!music.openFromFile("libs/audio/music.ogg")) {
00065     std::cout « "Error loading music."
00066               « "\n";
00067   }
00068   music.setRelativeToListener(true);
00069   music.setVolume(20);
00070   music.setLoop(true);
00071   music.play();
00072
00073   // Sounds
00074   sf::SoundBuffer punchbuf;
00075   punchbuf.loadFromFile("libs/audio/punch.ogg");
00076   sf::Sound punch;
00077   punch.setBuffer(punchbuf);
00078
00079   sf::SoundBuffer deathbuf;
00080   deathbuf.loadFromFile("libs/audio/death.ogg");
00081   sf::Sound death;
00082   death.setBuffer(deathbuf);
00083
00084   sf::SoundBuffer shootbuf;
00085   shootbuf.loadFromFile("libs/audio/shot.ogg");
00086   sf::Sound shoot;
00087   shoot.setBuffer(shootbuf);
00088
00089   sf::SoundBuffer drinkbuf;
00090   drinkbuf.loadFromFile("libs/audio/drink.ogg");
00091   sf::Sound drink;
00092   drink.setBuffer(drinkbuf);
00093
00094   sf::SoundBuffer pickbuf;
00095   pickbuf.loadFromFile("libs/audio/pickup.ogg");
00096   sf::Sound pickup;
00097   pickup.setBuffer(pickbuf);
00098
00099   // Textures
```

```
00100   sf::Texture* walltexture = new sf::Texture();
00101   walltexture->loadFromFile("libs/images/wall_left.png");
00102
00103   // sf::Texture enemytexture;
00104   // sf::IntRect enemytexture_crop(64, 11, 17, 21);
00105   // enemytexture.loadFromFile("libs/images/pumpkin_dude.png",
00106   // enemytexture_crop);
00107   Animation enemyAnimation("libs/images/pumpkin_dude.png", 16, 23);
00108   // sf::Texture enemytexture2;
00109   // sf::IntRect enemytexture_crop2(64, 11, 17, 21);
00110   // enemytexture2.loadFromFile("libs/images/skeleton_dude.png",
00111   // enemytexture_crop2);
00112   Animation enemyAnimation2("libs/images/skeleton_dude.png", 16, 20);
00113
00114   Animation boss_animation("libs/images/boss.png", 32, 36);
00115
00116   Texture weapon_texture;
00117   weapon_texture.loadFromFile("libs/images/simple gun.png");
00118   Texture projectile_texture;
00119   projectile_texture.loadFromFile("libs/images/simple bullet.png");
00120   Texture enemy_projectile_texture;
00121   enemy_projectile_texture.loadFromFile("libs/images/enemy bullet.png");
00122   sf::Texture playerTexture;
00123   playerTexture.loadFromFile("libs/images/character.png");
00124   sf::Texture* heart_texture = new sf::Texture();
00125   heart_texture->loadFromFile("libs/images/ui_heart_full.png");
00126   sf::Texture* heart_half_texture = new sf::Texture();
00127   heart_half_texture->loadFromFile("libs/images/ui_heart_half.png");
00128   sf::Texture* heart_empty_texture = new sf::Texture();
00129   heart_empty_texture->loadFromFile("libs/images/ui_heart_empty.png");
00130   sf::Texture* backgroundtexture = new sf::Texture();
00131   backgroundtexture->loadFromFile("libs/images/floor_1.png");
00132   sf::Texture player_death_texture;
00133   player_death_texture.loadFromFile("libs/images/character_grave.png");
00134
00135   sf::Texture healPotTexture;
00136   healPotTexture.loadFromFile("libs/images/flask_big_red.png");
00137   sf::Texture speedPotTexture;
00138   speedPotTexture.loadFromFile("libs/images/flask_big_blue.png");
00139   sf::Texture coinTexture;
00140   coinTexture.loadFromFile("libs/images/coin.png");
00141   sf::Texture* healPotTexture2 = new sf::Texture();
00142   healPotTexture2->loadFromFile("libs/images/flask_big_red.png");
00143   sf::Texture* speedPotTexture2 = new sf::Texture();
00144   speedPotTexture2->loadFromFile("libs/images/flask_big_blue.png");
00145   sf::Texture* coinTexture2 = new sf::Texture();
00146   coinTexture2->loadFromFile("libs/images/coin.png");
00147   std::vector<sf::Texture*> item_textures = {healPotTexture2, speedPotTexture2,
00148                                              coinTexture2};
00149   std::vector<sf::Texture*> heart_textures = {heart_texture, heart_half_texture,
00150                                               heart_empty_texture};
00151   Menu menu(font, walltexture);
00152
00153   // Generate map and player
00154   HUD hud(font, heart_textures, backgroundtexture, backgroundtexture,
00155           item_textures);
00156   // Map map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00157   // senemy_projectile_texture);
00158
00159   std::vector<Projectile> active_projectiles;  // Keep track of projectiles that
00160                                                // exist inside the SFML window.
00161
00162   // Generate map and player
00163   Map map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00164           &enemy_projectile_texture);
00165
00166   Weapon player_starting_weapon = Weapon(&weapon_texture, &projectile_texture);
00167
00168   float player_speed = 200;
00169   Vector2f spawnPos = Vector2f(100.0f, 100.0f);
00170   Player player(&playerTexture, &player_death_texture, spawnPos, player_speed,
00171                 &player_starting_weapon);
00172
00173   Clock clock;
00174   int resetCounter = 1;
00175   // This while loop runs as long as window is open. In this case it runs until
00176   // window.close(); is called.
00177   while (window.isOpen()) {
00178     Event event;
00179     if (menu.active_) {
00180       while (window.pollEvent(event)) {
00181         if (event.type == Event::Closed) {
00182           window.close();
00183         }  // This event closes the window when close button is pressed.
00184       }
00185       resetCounter = 0;
00186       window.clear();
```

```
00187       menu.Display(window);
00188       window.display();
00189     } else if (resetCounter == 0) {
00190       srand(clock.restart().asMilliseconds());
00191       map = Map(walltexture, &enemyAnimation, &enemyAnimation2, &boss_animation,
00192                 &enemy_projectile_texture);
00193       player = Player(&playerTexture, &player_death_texture, spawnPos,
00194                       player_speed, &player_starting_weapon);
00195       active_projectiles.clear();
00196       resetCounter++;
00197     } else {
00198       float time = clock.restart().asSeconds();
00199
00200       // This while loop goes through all window events, such as key presses
00201       // and mouse presses.
00202
00203       while (window.pollEvent(event)) {
00204         if (event.type == Event::Closed) {
00205           window.close();
00206         }  // This event closes the window when close button is pressed.
00207       }
00208
00209       std::vector<int> sounds = player.Update(time, map, active_projectiles);
00210       if (sounds[0]) {
00211         shoot.play();
00212       }
00213       if (sounds[1]) {
00214         punch.play();
00215       }
00216       if (sounds[2]) {
00217         drink.play();
00218       }
00219       if (sounds[3]) {
00220         pickup.play();
00221       }
00222       // update enemies and remove them if they are dead
00223       for (auto& enemy : map.enemies) {
00224         if (enemy->isRanged()) {
00225           enemy->Update2(time, player.getPosition(), active_projectiles);
00226         } else {
00227           enemy->Update(time, player.getPosition());
00228         }
00229         for (unsigned int i = 0; i < map.enemies.size(); i++) {
00230           if (!map.enemies[i]->alive_) {
00231             map.SpawnItem(i, &healPotTexture, &speedPotTexture, &coinTexture);
00232             map.enemies.erase(map.enemies.begin() + i);
00233             death.play();
00234           }
00235         }
00236       }
00237
00238       // update active projectiles and remove them if they have hit a wall or
00239       // an enemy
00240       for (auto& proj : active_projectiles) {
00241         // sf::Vector2u secret_size = proj.projectile_texture_->getSize();
00242         // std::cout « secret_size.x « " " « secret_size.y « "\n";
00243         if (proj.Update(time, map, player)) {
00244           proj.deActivate();
00245           for (unsigned int i = 0; i < active_projectiles.size(); i++) {
00246             if (!active_projectiles[i].active_) {
00247               active_projectiles.erase(active_projectiles.begin() + i);
00248             }
00249           }
00250         }
00251       }
00252       // std::cout « active_projectiles.size() « "\n";
00253
00254       window.clear();
00255       Collider playerCollider = player.getCollider();
00256       map.Display(window, playerCollider);
00257       player.Draw(window);
00258
00259       for (auto& enemy : map.enemies) {
00260         enemy->Draw(window);
00261       }
00262       for (auto& proj : active_projectiles) {
00263         proj.Draw(window);
00264       }
00265
00266       hud.Display(window);
00267       hud.Update(player.items_, player.hp_);
00268
00269       if (player.isAlive() == false) {
00270         // window.draw(text);
00271         // window.clear();
00272         menu.active_ = true;
00273       }
```

```
00274       window.display();
00275     }
00276   }
00277
00278   // for (auto& enemy : map.enemies) {
00279   //   delete enemy;
00280   // }
00281   delete walltexture;
00282   delete backgroundtexture;
00283   return 0;
00284 }
```

6.23 src/map.cpp File Reference

```
#include "map.hpp"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Keyboard.hpp>
#include <cstddef>
#include <iostream>
#include <map>
#include <numeric>
#include <utility>
#include "projectile.hpp"
```

Include dependency graph for map.cpp:

6.24 map.cpp

```
00001 #include "map.hpp"
00002
00003 #include <stdio.h>  /* printf, scanf, puts, NULL */
00004 #include <stdlib.h> /* srand, rand */
00005 #include <time.h>   /* time */
00006
00007 #include <SFML/System/Vector2.hpp>
00008 #include <SFML/Window/Keyboard.hpp>
00009 #include <cstddef>
00010 #include <iostream>
00011 #include <map>
00012 #include <numeric>
00013 #include <utility>
00014
00015 #include "projectile.hpp"
00016 //#include <cmath> /* abs */
00017
00018 Map::Map(sf::Texture *wall_texture, Animation *enemy_animation,
00019          Animation *enemy_animation2, Animation *boss_animation,
00020          sf::Texture *enemy_projectile_texture) {
00021   std::vector<std::pair<int, int» visited;
00022   visited.push_back({100, 100});
00023   int depth = 0;
00024   Generate(wall_texture, enemy_animation, enemy_animation2, boss_animation,
00025            enemy_projectile_texture, 0, 0, {0, 1, 1, 0, 4}, visited);
00026
00027   int highest_depth = 0;
00028   unsigned int highest_depth_index = 0;
00029   for (unsigned int i = 0; i < rooms.size(); i++) {
00030     if (rooms[i].depth_ > highest_depth) {
00031       highest_depth = rooms[i].depth_;
00032       highest_depth_index = i;
00033     }
00034   }
00035   sf::Vector2f boss_spawn_pos(rooms[highest_depth_index].xBound1,
00036                               rooms[highest_depth_index].yBound1);
00037   sf::Vector2f enemy_spawn_pos;
00038   enemy_spawn_pos.x = boss_spawn_pos.x + 50 + (rand() % 350);
00039   enemy_spawn_pos.y = boss_spawn_pos.y + 50 + (rand() % 350);
00040
00041   FinalBoss *new_enemy = new FinalBoss(boss_animation, enemy_spawn_pos, 75,
```

```
00042                                       enemy_projectile_texture);
00043   // std::unique_ptr<Enemy> new_enemy(new FinalBoss(boss_animation,
00044   // enemy_spawn_pos, 75));
00045   enemies.push_back(new_enemy);
00046 };
00047
00048 bool Map::Generate(sf::Texture *wall_texture, Animation *enemy_animation,
00049                    Animation *enemy_animation2, Animation *boss_animation,
00050                    sf::Texture *enemy_projectile_texture, int x, int y,
00051                    std::vector<int> openings,
00052                    std::vector<std::pair<int, int» &visited) {
00053   static int calls = 0;
00054   calls++;
00055   bool vi = false;
00056   for (auto m : visited) {
00057     if (x == m.first && y == m.second) {
00058       vi = true;
00059     }
00060   }
00061
00062   if (x < mapSize.x && y < mapSize.y && x >= 0 && y >= 0 &&
00063       std::accumulate(openings.begin(), openings.end() - 1, 0) > 1 &&
00064       vi == false) {
00065     visited.push_back({x, y});
00066     std::vector<int> new_openings;
00067     for (unsigned int i = 0; i < openings.size() - 1; i++) {
00068       if (openings[i] == 1 && openings[4] != int(i)) {
00069         new_openings.clear();
00070         for (auto j = 0; j < 4; j++) {
00071           new_openings.push_back(rand() % 2);
00072         }
00073         if (i < 2) {
00074           new_openings[i + 2] = 1;
00075           new_openings.push_back(i + 2);
00076         } else {
00077           new_openings[i - 2] = 1;
00078           new_openings.push_back(i - 2);
00079         }
00080         if (i == 0) {
00081           bool success = Generate(
00082               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00083               enemy_projectile_texture, x, y - 1, new_openings, visited);
00084           if (success == false) {
00085             openings[0] = 0;
00086           }
00087         } else if (i == 1) {
00088           bool success = Generate(
00089               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00090               enemy_projectile_texture, x + 1, y, new_openings, visited);
00091           if (success == false) {
00092             openings[1] = 0;
00093           }
00094         } else if (i == 2) {
00095           bool success = Generate(
00096               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00097               enemy_projectile_texture, x, y + 1, new_openings, visited);
00098           if (success == false) {
00099             openings[2] = 0;
00100           }
00101         } else {
00102           bool success = Generate(
00103               wall_texture, enemy_animation, enemy_animation2, boss_animation,
00104               enemy_projectile_texture, x - 1, y, new_openings, visited);
00105           if (success == false) {
00106             openings[3] = 0;
00107           }
00108         }
00109       }
00110     }
00111
00112     /*
00113     The enemies are added inside the map and they are going to be located with
00114     respect to the room spawn positions. That means that every room is going to
00115     have a random amount of enemies located at random coordinates in the room,
00116     but the enemy location is not limited to the rooms, because enemies are
00117     located in the map, not in the rooms.
00118     */
00119     sf::Vector2f room_spawn_pos =
00120         sf::Vector2f(25.0f + 50 * 10 * x, 25.0f + 50 * 10 * y);
00121     // int random_enemy_amount = std::abs(rand() % 5);
00122     // int random_enemy_amount = rand() % 5;
00123     // int random_enemy_amount2 = rand() % 3;
00124     int random_enemy_amount = rand() % 4;
00125     int random_enemy_amount2 = rand() % 2;
00126     // int random_enemy_amount2 = 0;
00127     for (int i = 0; i < random_enemy_amount; i++) {
00128       sf::Vector2f enemy_spawn_pos;
```

```
00129       enemy_spawn_pos.x = room_spawn_pos.x + 50 + (rand() % 350);
00130       enemy_spawn_pos.y = room_spawn_pos.y + 50 + (rand() % 350);
00131       // std::unique_ptr<Enemy> new_enemy =
00132       // std::make_unique<ChasingEnemy>(enemy_animation, enemy_spawn_pos, 100);
00133       // ChasingEnemy new_enemy(enemy_animation, enemy_spawn_pos, 100);
00134       ChasingEnemy *new_enemy =
00135           new ChasingEnemy(enemy_animation, enemy_spawn_pos, 100);
00136       enemies.push_back(new_enemy);
00137     }
00138     for (int i = 0; i < random_enemy_amount2; i++) {
00139       sf::Vector2f enemy_spawn_pos;
00140       enemy_spawn_pos.x = room_spawn_pos.x + 50 + (rand() % 350);
00141       enemy_spawn_pos.y = room_spawn_pos.y + 50 + (rand() % 350);
00142       // std::unique_ptr<Enemy> new_enemy =
00143       // std::make_unique<ChasingEnemy>(enemy_animation, enemy_spawn_pos, 75);
00144       // std::unique_ptr<Enemy> new_enemy(new RangedEnemy(enemy_animation2,
00145       // enemy_spawn_pos, 75, enemy_projectile_texture)); RangedEnemy
00146       // new_enemy(enemy_animation2, enemy_spawn_pos, 75);
00147       RangedEnemy *new_enemy = new RangedEnemy(
00148           enemy_animation2, enemy_spawn_pos, 75, enemy_projectile_texture);
00149       enemies.push_back(new_enemy);
00150     }
00151
00152     Room room(wall_texture,
00153               sf::Vector2f(25.0f + 50 * 10 * x, 25.0f + 50 * 10 * y), openings,
00154               calls);
00155     rooms.push_back(room);
00156     calls--;
00157     return true;
00158   }
00159   calls--;
00160   return false;
00161 };
00162
00163 void Map::Display(sf::RenderWindow &window, Collider playerCollider) {
00164   for (auto r : rooms) {
00165     r.Display(window, playerCollider, enemies);
00166   }
00167   for (auto i : items) {
00168     i.Draw(window);
00169   }
00170 };
00171
00172 void Map::NextRoom(int direction) {
00173   for (unsigned int i = 0; i < rooms.size(); i++) {
00174     rooms[i].MoveRoom(direction, items);
00175   }
00176   for (unsigned int i = 0; i < enemies.size(); i++) {
00177     enemies[i]->MoveRoom(direction);
00178   }
00179 }
00180
00181 void Map::SpawnItem(unsigned int &i, sf::Texture *hp_text,
00182                     sf::Texture *speed_text, sf::Texture *coin_text) {
00183   sf::Vector2f pos = enemies[i]->getPosition();
00184   int item_index = rand() % 3;
00185   if (item_index == 0) {
00186     Item drop(hp_text, pos, "hp_pot");
00187     items.push_back(drop);
00188   } else if (item_index == 1) {
00189     Item drop(speed_text, pos, "speed_pot");
00190     items.push_back(drop);
00191   } else {
00192     Item drop(coin_text, pos, "coin");
00193     items.push_back(drop);
00194   }
00195 }
00196
00197 void Map::removeItem(std::vector<Item>::iterator i) { items.erase(i); };
```

6.25 src/map.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <memory>
#include <vector>
```

```
#include "animation.hpp"
#include "collider.hpp"
#include "item.hpp"
#include "room.hpp"
```

Include dependency graph for map.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Map

6.26 map.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <memory>
00006 #include <vector>
00007
00008 #include "animation.hpp"
00009 #include "collider.hpp"
00010 #include "item.hpp"
00011 #include "room.hpp"
00012
00013 #ifndef MAP_CLASS
00014 #define MAP_CLASS
00015
00016 class Projectile;  // Forward declaration is used here to avoid circular
00017                    // dependency with projectile class
00018 /**Class for the map*/
00019 class Map {
00020  public:
00021   /**Constructor for map*/
00022   Map(sf::Texture* wall_texture, Animation* enemy_animation,
00023       Animation* enemy_animation2, Animation* boss_animation,
00024       sf::Texture* enemy_projectile_texture);
00025   /**Generates random dungeon and spawns random amount of enemies to each
00026   /*room.*/
00027   bool Generate(sf::Texture* wall_texture, Animation* enemy_animation,
00028                 Animation* enemy_animation2, Animation* boss_animation,
00029                 sf::Texture* enemy_projectile_texture, int x, int y,
00030                 std::vector<int> openings,
00031                 std::vector<std::pair<int, int>>& visited);
00032   /**Used for rendering the map*/
00033   void Display(sf::RenderWindow& window, Collider playerCollider);
00034   /**Used for whern player traverses between rooms.*/
00035   void NextRoom(int direction);
00036   /**Used for spawning random item*/
00037   void SpawnItem(unsigned int& i, sf::Texture* hp_text, sf::Texture* speed_text,
00038                  sf::Texture* coin_text);
00039   /**Used for picking up items*/
00040   void removeItem(std::vector<Item>::iterator i);
00041
00042   std::vector<Room> rooms;
00043   // std::vector<unique_ptr<Enemy>> enemies;
00044   // std::vector<Enemy> enemies;
00045   std::vector<Enemy*> enemies;
00046   std::vector<std::vector<Room>> layout;
00047   sf::Vector2f mapSize = sf::Vector2f(6.0f, 6.0f);
00048   std::vector<Item> items;
00049 };
00050
00051 #endif
```

6.27 src/menu.cpp File Reference

```
#include "menu.hpp"
#include <SFML/Graphics/Color.hpp>
#include <SFML/Graphics/Font.hpp>
#include <SFML/Graphics/RectangleShape.hpp>
```

```
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/System/Vector2.hpp>
#include <SFML/Window/Mouse.hpp>
#include <iostream>
```

Include dependency graph for menu.cpp:

6.28 menu.cpp

```
00001 #include "menu.hpp"
00002
00003 #include <SFML/Graphics/Color.hpp>
00004 #include <SFML/Graphics/Font.hpp>
00005 #include <SFML/Graphics/RectangleShape.hpp>
00006 #include <SFML/Graphics/RenderWindow.hpp>
00007 #include <SFML/System/Vector2.hpp>
00008 #include <SFML/Window/Mouse.hpp>
00009 #include <iostream>
00010
00011 Menu::Menu(sf::Font font, sf::Texture* menubackground)
00012     : font_(font), active_(true) {
00013   background_.setSize(sf::Vector2f(500, 650));
00014   background_.setPosition(sf::Vector2f(0, 0));
00015   background_.setTexture(menubackground);
00016   sf::Text StartText;
00017   StartText.setString("Start New Game");
00018   StartText.setCharacterSize(36);
00019   StartText.setPosition(50, 300);
00020   StartText.setFillColor(sf::Color::Yellow);
00021   StartText.setFont(font_);
00022   texts_.push_back(StartText);
00023   sf::Text QuitText;
00024   QuitText.setString("Quit Game");
00025   QuitText.setCharacterSize(36);
00026   QuitText.setPosition(50, 350);
00027   QuitText.setFillColor(sf::Color::White);
00028   QuitText.setFont(font_);
00029   texts_.push_back(QuitText);
00030 }
00031 void Menu::Display(sf::RenderWindow& window) {
00032   if (active_) {
00033     this->Update(window);
00034     window.draw(background_);
00035     for (unsigned int i = 0; i < texts_.size(); i++) {
00036       window.draw(texts_[i]);
00037     }
00038   }
00039 }
00040 Menu::~Menu(){};
00041
00042 void Menu::Update(sf::RenderWindow& relativeTo) {
00043   if (sf::Mouse::isButtonPressed(sf::Mouse::Left) and active_) {
00044     sf::Vector2i pos = sf::Mouse::getPosition(relativeTo);
00045     if (pos.y > 30 && pos.y < (300 + 36)) {
00046       active_ = false;
00047     }
00048     if (pos.y > (350) && pos.y < (350 + 36)) {
00049       relativeTo.close();
00050     }
00051   }
00052 }
```

6.29 src/menu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Text.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <vector>
```

Include dependency graph for menu.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Menu

6.30 menu.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Text.hpp>
00004 #include <SFML/Graphics/Texture.hpp>
00005 #include <SFML/System/Vector2.hpp>
00006 #include <vector>
00007
00008 // include "item.hpp"
00009
00010 #ifndef MENU_CLASS
00011 #define MENU_CLASS
00012 /**Class for main menu*/
00013 class Menu {
00014  public:
00015   /**Constructor for menu*/
00016   Menu(sf::Font font, sf::Texture* menubackground);
00017   /**Default destructor*/
00018   ~Menu();
00019   /**Renders the main menu*/
00020   void Display(sf::RenderWindow& window);
00021   /**Polls for actions in main menu*/
00022   void Update(sf::RenderWindow& window);
00023   sf::RectangleShape background_;
00024   sf::Font font_;
00025   std::vector<sf::Text> texts_;
00026   bool active_;
00027 };
00028 #endif
```

6.31 src/player.cpp File Reference

```
#include "player.hpp"
#include <SFML/Audio/Sound.hpp>
#include <SFML/System/Vector2.hpp>
#include <iostream>
```

Include dependency graph for player.cpp:

6.32 player.cpp

```
00001 #include "player.hpp"
00002
00003 #include <SFML/Audio/Sound.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <iostream>
00006
00007 Player::Player(sf::Texture* texture, sf::Texture* dead_texture,
00008                sf::Vector2f spawnPos, float speed, Weapon* starting_weapon)
00009     : speed_(speed) {
00010   death_texture_ = dead_texture;
00011   player_weapons_.push_back(*starting_weapon);
00012   body_.setSize(sf::Vector2f(40.0f, 40.0f));
00013   body_.setOrigin(body_.getSize() / 2.0f);
00014   body_.setPosition(spawnPos);
00015   body_.setTexture(texture);
00016 };
00017
00018 Player::~Player(){};
00019
00020 void Player::Draw(sf::RenderWindow& window) { window.draw(body_); };
00021
00022 sf::Vector2f Player::getPosition() { return body_.getPosition(); };
00023
00024 bool Player::isAlive() { return alive_; };
00025
```

```
00026 float Player::getHp() { return hp_; }
00027
00028 float Player::getShield() { return shield_; }
00029
00030 bool Player::useItem() {
00031   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num1)) {
00032     auto i = items_.begin();
00033     while (i != items_.end()) {
00034       if (i->type == "hp_pot" && itemTimer > 0.5) {
00035         if (hp_ < 6) {
00036           hp_ += 1;
00037           i = items_.erase(i);
00038           itemTimer = 0;
00039           return true;
00040         } else {
00041           return false;
00042         }
00043       } else {
00044         ++i;
00045       }
00046     }
00047   }
00048
00049   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Num2)) {
00050     auto i = items_.begin();
00051     while (i != items_.end()) {
00052       if (i->type == "speed_pot" && itemTimer > 0.5) {
00053         speed_ += 10;
00054         speedTimer = 15;
00055         i = items_.erase(i);
00056         itemTimer = 0;
00057         return true;
00058       } else {
00059         ++i;
00060       }
00061     }
00062   }
00063 }
00064
00065 void Player::Damage(float damage) {
00066   if (shield_ > 0) {
00067     shield_ -= damage;
00068     if (shield_ < 0) shield_ = 0;
00069   } else
00070     hp_ -= damage;
00071   if (hp_ <= 0) {
00072     std::cout « "The player has died!" « std::endl;
00073     alive_ = false;
00074   }
00075 }
00076
00077 std::vector<int> Player::Update(float time, Map& map,
00078                                 std::vector<Projectile>& active_projectiles) {
00079   std::vector<int> ret = {0, 0, 0, 0};
00080   if (alive_ == false) {
00081     if (dead_ == false) {
00082       sf::Vector2u texture_size = death_texture_->getSize();
00083       sf::Vector2f texture_size_f(float(texture_size.x), float(texture_size.y));
00084       body_.setSize(texture_size_f);
00085       body_.setOrigin(body_.getSize() / 2.0f);
00086       sf::Vector2f temp_position = body_.getPosition();
00087       std::cout « "Death position: " « temp_position.x « " "
00088                 « temp_position.y « std::endl;
00089       body_.setPosition(temp_position);
00090       body_.setTexture(death_texture_);
00091       dead_ = true;
00092     }
00093     // return active_projectiles;
00094   }
00095   if (this->useItem()) {
00096     ret[2] = 1;
00097   }
00098   currentSpeed_ = speed_;
00099   actionTimer_ += time;
00100   shot_ += time;  // cooldown for firing weapon
00101   itemTimer += time;
00102   sf::Vector2f movement(0.0f, 0.0f);
00103   float firerate = player_weapons_[chosen_weapon_].fire_rate_;
00104
00105   if (speedTimer > 0) {
00106     speedTimer -= time;
00107   } else if (speedTimer < 0) {
00108     speedTimer = 0;
00109     speed_ -= 10;
00110   }
00111
00112   if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
```

```
00113     movement.y -= currentSpeed_ * time;
00114     direction_ = movement;
00115   }
00116
00117   if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
00118     movement.x -= currentSpeed_ * time;
00119     direction_ = movement;
00120   }
00121
00122   if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
00123     movement.y += currentSpeed_ * time;
00124     direction_ = movement;
00125   }
00126
00127   if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
00128     movement.x += currentSpeed_ * time;
00129     direction_ = movement;
00130   }
00131
00132   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) &&
00133       shot_ >= firerate) {  // Up arrow key is used to shoot projectiles.
00134     sf::Vector2f direction = sf::Vector2f(0.0f, -10.0f);
00135     Projectile new_projectile = this->UseWeapon(direction);
00136     active_projectiles.push_back(new_projectile);
00137     shot_ = 0.0;
00138     ret[0] = 1;
00139   }
00140   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) &&
00141       shot_ >= firerate) {  // Down arrow key is used to shoot projectiles.
00142     sf::Vector2f direction = sf::Vector2f(0.0f, 10.0f);
00143     Projectile new_projectile = this->UseWeapon(direction);
00144     active_projectiles.push_back(new_projectile);
00145     shot_ = 0.0;
00146     ret[0] = 1;
00147   }
00148   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) &&
00149       shot_ >= firerate) {  // Left arrow key is used to shoot projectiles.
00150     sf::Vector2f direction = sf::Vector2f(-10.0f, 0.0f);
00151     Projectile new_projectile = this->UseWeapon(direction);
00152     active_projectiles.push_back(new_projectile);
00153     shot_ = 0.0;
00154     ret[0] = 1;
00155   }
00156   if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) &&
00157       shot_ >= firerate) {  // Right arrow key is used to shoot projectiles.
00158     sf::Vector2f direction = sf::Vector2f(10.0f, 0.0f);
00159     Projectile new_projectile = this->UseWeapon(direction);
00160     active_projectiles.push_back(new_projectile);
00161     shot_ = 0.0;
00162     ret[0] = 1;
00163   }
00164
00165   Traverse(map);
00166   body_.move(movement);
00167
00168   // check that player is not colliding with any of the enemies and take damage
00169   // if colliding
00170   damaged_ += time;
00171   Collider collider = this->getCollider();
00172   for (auto& enemy : map.enemies) {
00173     Collider EnemyCollider = enemy->getCollider();
00174     if (collider.checkCollider(EnemyCollider, 0.5) && damaged_ >= 1.5) {
00175       this->Damage(enemy->damage_);
00176       damaged_ = 0.0;
00177       ret[1] = 1;
00178     }
00179   }
00180   for (unsigned int i = 0; i < map.items.size(); i++) {
00181     Collider itemCollider = map.items[i].getCollider();
00182     if (collider.checkCollider(itemCollider, 0)) {
00183       items_.push_back(map.items[i]);
00184       map.items.erase(map.items.begin() + i);
00185       ret[3] = 1;
00186     }
00187   }
00188   return ret;
00189 };
00190
00191 Projectile Player::UseWeapon(sf::Vector2f& direction) {
00192   return player_weapons_[chosen_weapon_].Fire(body_.getPosition(), direction);
00193 };
00194
00195 void Player::Traverse(Map& map) {
00196   sf::Vector2f pos = body_.getPosition();
00197
00198   if (pos.x < 25) {
00199     map.NextRoom(1);
```

```
00200     body_.move(sf::Vector2f(500, 0));
00201   }
00202   if (pos.x > 25 + 10 * 50) {
00203     map.NextRoom(3);
00204     body_.move(sf::Vector2f(-500, 0));
00205   }
00206   if (pos.y < 25) {
00207     map.NextRoom(2);
00208     body_.move(sf::Vector2f(0, 500));
00209   }
00210   if (pos.y > 25 + 10 * 50) {
00211     map.NextRoom(0);
00212     body_.move(sf::Vector2f(0, -500));
00213   }
00214 }
```

6.33 src/player.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <vector>
#include "item.hpp"
#include "map.hpp"
#include "projectile.hpp"
#include "weapon.hpp"
```

Include dependency graph for player.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Player

6.34 player.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005 #include <vector>
00006
00007 #include "item.hpp"
00008 #include "map.hpp"
00009 #include "projectile.hpp"
00010 #include "weapon.hpp"
00011
00012 #ifndef PLAYER_CLASS
00013 #define PLAYER_CLASS
00014 /**Class for the player object*/
00015 class Player {
00016  public:
00017   /**Constructor for player*/
00018   Player(sf::Texture* texture, sf::Texture* dead_texture, sf::Vector2f spawnPos,
00019          float speed, Weapon* starting_weapon);
00020   /**Default constructor*/
00021   ~Player();
00022   /**Renders the player*/
00023   void Draw(sf::RenderWindow& window);
00024
00025   // void GiveWeapon(Weapon weapon) {player_weapon_ = weapon;};
00026   /**Returns player position*/
00027   sf::Vector2f getPosition();
00028   /**Return alive*/
00029   bool isAlive();
00030   /**Return healh*/
00031   float getHp();
00032
00033   float getShield();
00034   /**Uses item from inventory*/
```

```
00035   bool useItem();
00036   /**Decrease health*/
00037   void Damage(float dmg);
00038   /**Update attributes*/
00039   std::vector<int> Update(float time, Map& map, std::vector<Projectile>&);
00040   /**Creates projectile*/
00041   Projectile UseWeapon(sf::Vector2f&);
00042   /**Moves the map when player traverses between rooms*/
00043   void Traverse(Map& map);
00044   /**Returns the player collider*/
00045   Collider getCollider() { return Collider(body_); }
00046
00047   float speed_;
00048   float hp_ = 6;
00049   float shield_ = 0;
00050   float currentSpeed_;
00051   float actionTimer_;
00052   float shot_ = 0.0;
00053   float damaged_ = 0.0;
00054   int chosen_weapon_ = 0;
00055   float speedTimer = 0;
00056   float itemTimer = 0;
00057   bool alive_ = true;
00058   bool dead_ = false;
00059   sf::Texture* death_texture_;
00060   std::vector<Weapon> player_weapons_;
00061   std::vector<Item> items_;
00062   sf::RectangleShape body_;
00063   sf::Vector2f direction_;
00064 };
00065
00066 #endif
```

6.35 src/projectile.cpp File Reference

```
#include "projectile.hpp"
#include "player.hpp"
```

Include dependency graph for projectile.cpp:

6.36 projectile.cpp

```
00001 #include "projectile.hpp"
00002 #include "player.hpp"
00003
00004 //#include <iostream>
00005
00006 /*
00007 * Constructor of projectile, texture is the texture of the projectile.
00008 */
00009 Projectile::Projectile(sf::Texture* texture) {
00010   //friendly_ = true;
00011   //friendly_.push_back(1);
00012   projectile_texture_ = texture;
00013 };
00014
00015 //Projectile::Projectile(sf::Texture* texture, bool friendly)
00016 //: friendly_(friendly), projectile_texture_(texture) {};
00017
00018 Projectile::~Projectile() {};
00019
00020 /*
00021 * Method for drawing the projectile
00022 */
00023 void Projectile::Draw(sf::RenderWindow& window) { window.draw(body_); };
00024
00025 /*
00026 * Method for updating the projectile location; moves the projectile towards its trajectory
00027 * Checks if projectile hits wall or enemy
00028 */
00029 bool Projectile::Update(float time, Map& map, Player& player) {
00030   actionTimer_ += time;
00031   if(actionTimer_ > 1.0) {
00032     return 1;
00033   }
00034   body_.move(projectile_trajectory_);
00035   Collider col = this->getCollider();
```

```
00036   //std::cout « friendly_;
00037   //if(friendly_ == 1) {
00038
00039     //secret_size is used for determining who the projectile harms, because otherwise there will be
      memory issues.
00040   sf::Vector2u secret_size = this->projectile_texture_->getSize();
00041   if(secret_size.x < 500) { //Enemy projectile textures happen to be, by coincidence, larger than
      player projectile textures
00042     for (auto& enemy : map.enemies) {
00043     Collider enemyCollider = enemy->getCollider();
00044     if (col.checkCollider(enemyCollider, 0)) {
00045         enemy->Damage(15);
00046         return 1;
00047     }
00048     }
00049   } else {
00050     Collider playerCollider = player.getCollider();
00051     if (col.checkCollider(playerCollider, 0)) {
00052         player.Damage(2);
00053         return 1;
00054     }
00055   }
00056   for (auto room : map.rooms) {
00057     for (auto wall : room.walls) {
00058       for (auto w : wall) {
00059         if (w.type_ == 1) {
00060           Collider wallCollider = w.getCollider();
00061           if (wallCollider.checkCollider(col, 1)) {
00062             return 1;
00063           }
00064         }
00065       }
00066     }
00067   }
00068   return 0;
00069   //std::cout « projectile_trajectory_.x « "\t" « projectile_trajectory_.y « "\n";
00070   //std::cout « body_.getPosition().x « "\t" « body_.getPosition().y « "\n";
00071 };
00072
00073 /*
00074 * Method for activating and deactivating the projectile. In main only active projectiles are drawn and
      updated on the SFML window.
00075 */
00076 void Projectile::activate() {active_ = true;};
00077 void Projectile::deActivate() {active_ = false;};
```

6.37 src/projectile.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <vector>
#include "map.hpp"
#include "enemy.hpp"
```

Include dependency graph for projectile.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Projectile

6.38 projectile.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <vector>
00005
00006 #include "map.hpp"
00007 #include "enemy.hpp"
00008
```

```
00009 #ifndef PROJECTILE_CLASS
00010 #define PROJECTILE_CLASS
00011
00012 class Player; //Forward declaration is used here to avoid circular dependency with player class
00013 /**Class for projectiles */
00014 class Projectile {
00015  public:
00016   /**Constructor*/
00017   Projectile(sf::Texture* texture);
00018   /**Default destructor*/
00019   ~Projectile();
00020   /**Renders the projectile*/
00021   void Draw(sf::RenderWindow& window);
00022   /**Updates attributes*/
00023   bool Update(float time, Map& map, Player& player);
00024   /**Changes active_ to true*/
00025   void activate();
00026   /**Changes active_ to false*/
00027   void deActivate();
00028   /**Returns the projectile collider*/
00029   Collider getCollider() {
00030     return Collider(body_);
00031   }
00032
00033   sf::Texture* projectile_texture_;
00034   sf::RectangleShape body_;
00035   sf::Vector2f projectile_trajectory_;
00036   bool active_ = false;
00037   float actionTimer_ = 0.0;
00038 };
00039
00040 #endif
```

6.39 src/readme.md File Reference

6.40 src/room.cpp File Reference

```
#include "room.hpp"
#include <math.h>
#include <SFML/Graphics/VertexArray.hpp>
#include <SFML/System/Vector2.hpp>
#include <cmath>
#include <iostream>
#include <ostream>
```

Include dependency graph for room.cpp:

6.41 room.cpp

```
00001 #include "room.hpp"
00002
00003 #include <math.h>
00004
00005 #include <SFML/Graphics/VertexArray.hpp>
00006 #include <SFML/System/Vector2.hpp>
00007 #include <cmath>
00008 #include <iostream>
00009 #include <ostream>
00010
00011 Room::Room(sf::Texture *texture, sf::Vector2f spawnPos,
00012            std::vector<int> openings, int depth)
00013     : active(false), depth_(depth) {
00014   xBound1 = spawnPos.x;
00015   xBound2 = spawnPos.x + maxSize.x * 50.0f;
00016
00017   yBound1 = spawnPos.y;
00018   yBound2 = spawnPos.y + maxSize.y * 50.0f;
00019
00020   sf::Texture *floortexture = new sf::Texture();
00021   floortexture->loadFromFile("libs/images/floor_1.png");
00022
```

```
00023   for (int x = 0; x < maxSize.x; x++) {
00024     walls.push_back(std::vector<Wall>());
00025
00026     for (int y = 0; y < maxSize.y; y++) {
00027       if (y == floor(maxSize.y / 2) &&
00028           ((x == 0 && openings[3] == 1) ||
00029            (x == maxSize.x - 1 && openings[1] == 1))) {
00030         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00031                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 2);
00032         walls[x].push_back(wall);
00033       } else if (x == floor(maxSize.x / 2) &&
00034                  ((y == 0 && openings[0] == 1) ||
00035                   (y == maxSize.y - 1 && openings[2] == 1))) {
00036         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00037                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 2);
00038         walls[x].push_back(wall);
00039       }
00040
00041       else if (x == 0 || x == maxSize.x - 1 || y == 0 || y == maxSize.y - 1) {
00042         Wall wall(texture, sf::Vector2f(50.0f, 50.f),
00043                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 1);
00044         walls[x].push_back(wall);
00045
00046       } else {
00047         Wall wall(floortexture, sf::Vector2f(50.0f, 50.f),
00048                   spawnPos + sf::Vector2f(50.0f * x, 50.0f * y), 0);
00049         walls[x].push_back(wall);
00050       }
00051     }
00052   }
00053 };
00054
00055 /*
00056  *Method for displaying individual walls
00057  *Checks collider with player and enemies
00058  */
00059 void Room::Display(sf::RenderWindow &window, Collider playerCollider,
00060                    std::vector<Enemy *> &enemies) {
00061   for (auto r : walls) {
00062     for (auto w : r) {
00063       w.Draw(window);
00064       Collider wallCollider = w.getCollider();
00065       if (w.type_ == 1) {
00066         wallCollider.checkCollider(playerCollider, 1);
00067
00068         for (auto i = 0; i < enemies.size(); i++) {
00069           Collider enemyCollider = enemies[i]->getCollider();
00070           wallCollider.checkCollider(enemyCollider, 1);
00071         }
00072       }
00073       if (w.type_ == 2) {
00074         for (auto i = 0; i < enemies.size(); i++) {
00075           Collider enemyCollider = enemies[i]->getCollider();
00076           wallCollider.checkCollider(enemyCollider, 1);
00077         }
00078       }
00079     }
00080   }
00081 };
00082
00083 // void Room::checkCollision(sf::RectangleShape &other_body) {
00084 //   for (unsigned int i = 0; i < walls.size(); i++) {
00085 //     for (unsigned int j = 0; j < walls[i].size(); j++) {
00086 //       if (walls[i][j].checkCollision(other_body) &&
00087 //           walls[i][j].body.getTexture() != nullptr) {
00088 //         float x = other_body.getPosition().x -
00089 //         walls[i][j].body.getPosition().x; float y =
00090 //         other_body.getPosition().y - walls[i][j].body.getPosition().y;
00091 //         other_body.move(sf::Vector2f(0.05 * x, 0.05 * y));
00092 //       }
00093 //     }
00094 //   }
00095 // }
00096
00097 void Room::MoveRoom(int direction, std::vector<Item> &items) {
00098   sf::Vector2f dir;
00099   if (direction == 0) {
00100     dir = sf::Vector2f(0, -10 * 50);
00101   }
00102   if (direction == 1) {
00103     dir = sf::Vector2f(10 * 50, 0);
00104   }
00105   if (direction == 2) {
00106     dir = sf::Vector2f(0, 10 * 50);
00107   }
00108   if (direction == 3) {
00109     dir = sf::Vector2f(-10 * 50, 0);
```

```
00110   }
00111
00112   for (auto i = 0; i < walls.size(); i++) {
00113     for (auto j = 0; j < walls[i].size(); j++) {
00114       walls[i][j].Move2(dir);
00115     }
00116   }
00117   for (auto i = 0; i < items.size(); i++) {
00118     items[i].body.move(dir);
00119   }
00120 }
00121
00122 void Room::Activate() { active = true; };
00123 void Room::Deactivate() { active = false; }
```

6.42 src/room.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include <vector>
#include "collider.hpp"
#include "enemy.hpp"
#include "item.hpp"
#include "wall.hpp"
```

Include dependency graph for room.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Room

6.43 room.hpp

```
00001 #ifndef ROOM_CLASS
00002 #define ROOM_CLASS
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006 #include <SFML/Graphics/Texture.hpp>
00007 #include <SFML/System/Vector2.hpp>
00008 #include <vector>
00009
00010 #include "collider.hpp"
00011 #include "enemy.hpp"
00012 #include "item.hpp"
00013 #include "wall.hpp"
00014 class Room {
00015  public:
00016   /*
00017    *Constructor for Room. Populates room edges with texture and floor with
00018    *nullptr walls
00019    */
00020   Room(sf::Texture* texture, sf::Vector2f spawnPos, std::vector<int> openings,
00021        int depth);
00022   /*
00023    *Method for displaying individual walls
00024    */
00025   void Display(sf::RenderWindow& window, Collider playerCollider,
00026                std::vector<Enemy*>& enemies);
00027   /**Checks collisions with walls*/
00028   void checkCollision(sf::RectangleShape& other_body);
00029   /**Moves the room when player traverses*/
00030   void MoveRoom(int direction, std::vector<Item>& items);
00031   /**Default destructor*/
00032   ~Room(){};
00033   void Activate();
00034   void Deactivate();
00035   bool active;
```

```
00036   sf::Vector2f maxSize = sf::Vector2f(10.0f, 10.0f);
00037   float xBound1;
00038   float xBound2;
00039   float yBound1;
00040   float yBound2;
00041   std::vector<std::vector<Wall» walls;
00042   int depth_;
00043 };
00044
00045 #endif
```

6.44 src/wall.cpp File Reference

```
#include "wall.hpp"
#include <SFML/System/Vector2.hpp>
```

Include dependency graph for wall.cpp:

6.45 wall.cpp

```
00001 #include "wall.hpp"
00002
00003 #include <SFML/System/Vector2.hpp>
00004
00005 Wall::Wall(sf::Texture *texture, sf::Vector2f size, sf::Vector2f position,
00006            int type) {
00007   body.setSize(size);
00008   body.setPosition(position);
00009   body.setTexture(texture);
00010   body.setOrigin(size / 2.0f);
00011   type_ = type;
00012 };
00013 void Wall::Draw(sf::RenderWindow &window) { window.draw(body); };
00014
00015 void Wall::Move2(sf::Vector2f dir) { body.move(dir); }
00016
00017 bool Wall::checkCollision(sf::RectangleShape other_body) {
00018   return body.getGlobalBounds().intersects(other_body.getGlobalBounds());
00019 }
```

6.46 src/wall.hpp File Reference

```
#include <SFML/Graphics/RectangleShape.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include "collider.hpp"
```

Include dependency graph for wall.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Wall

6.47 wall.hpp

```
00001 #ifndef WALL_CLASS
00002 #define WALL_CLASS
00003
00004 #include <SFML/Graphics/RectangleShape.hpp>
00005 #include <SFML/Graphics/RenderWindow.hpp>
00006 #include <SFML/Graphics/Texture.hpp>
00007 #include <SFML/System/Vector2.hpp>
00008
00009 #include "collider.hpp"
00010 /**Class for walls*/
00011 class Wall {
00012  public:
00013   /**Constructor*/
00014   Wall(sf::Texture* texture, sf::Vector2f size, sf::Vector2f position,
00015        int type);
00016   /**Renders the wall*/
00017   void Draw(sf::RenderWindow& window);
00018   /**Moves the wall*/
00019   void Move2(sf::Vector2f dir);
00020   /**Checks for collisions*/
00021   bool checkCollision(sf::RectangleShape other_body);
00022   /**Returns collider*/
00023   Collider getCollider() { return Collider(body); }
00024
00025   sf::RectangleShape body;
00026   int type_;
00027 };
00028
00029 #endif
```

6.48 src/weapon.cpp File Reference

`#include "weapon.hpp"`
Include dependency graph for weapon.cpp:

6.49 weapon.cpp

```
00001 #include "weapon.hpp"
00002
00003 /*
00004 * Constructor of weapon with textures for weapon displayed on ground/inventory and projectile texture
       that
00005 * is shown on the projectile when the weapon is fired.
00006 */
00007 Weapon::Weapon(sf::Texture* texture, sf::Texture* proj_texture) {
00008   weapon_body_.setTexture(texture);
00009   Projectile new_projectile = Projectile(proj_texture);
00010   weapon_projectile_.push_back(new_projectile);
00011 };
00012
00013 Weapon::~Weapon() {};
00014
00015 /*
00016 * Method for firing a single projectile that corresponds to the weapon.
00017 *
00018 * As the first parameter this method receives the position from where the weapon is being fired at,
00019 * the projectile will be located at this location initially.
00020 *
00021 * As the second parameter this method receives the trajectory for projectile,
00022 * it describes the speed and direction of the projectile.
00023 */
00024 Projectile Weapon::Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory) {
00025   Projectile proj(weapon_projectile_[0].projectile_texture_);
00026   proj.active_ = true;
00027   proj.body_.setTexture(proj.projectile_texture_);
00028   proj.body_.setOrigin(proj.body_.getSize() / 2.0f);
00029   proj.body_.setPosition(fire_position);
00030   proj.body_.setSize(sf::Vector2f(25.0f, 25.0f));
00031   proj.projectile_trajectory_ = fire_trajectory/* * proj.velocity_multiplier_*/;
00032   proj.body_.move(proj.projectile_trajectory_);
00033   return proj;
00034 };
```

6.50 src/weapon.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <SFML/Graphics/RenderWindow.hpp>
#include <SFML/Graphics/Texture.hpp>
#include <SFML/System/Vector2.hpp>
#include "projectile.hpp"
```

Include dependency graph for weapon.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Weapon

6.51 weapon.hpp

```
00001 #include <SFML/Graphics.hpp>
00002 #include <SFML/Graphics/RenderWindow.hpp>
00003 #include <SFML/Graphics/Texture.hpp>
00004 #include <SFML/System/Vector2.hpp>
00005
00006 #include "projectile.hpp"
00007
00008 #ifndef WEAPON_CLASS
00009 #define WEAPON_CLASS
00010 /**Class for weapon*/
00011 class Weapon {
00012  public:
00013   /**Constructor*/
00014   Weapon(sf::Texture* texture, sf::Texture* proj_texture);
00015   /**Default destructor*/
00016   ~Weapon();
00017
00018   /**Creates projectile*/
00019   Projectile Fire(sf::Vector2f fire_position, sf::Vector2f fire_trajectory);
00020
00021   sf::RectangleShape weapon_body_;
00022   //Projectile weapon_projectile_*;
00023   std::vector<Projectile> weapon_projectile_; //This vector is used instead of a pointer, because
      pointers give compile errors
00024   float fire_rate_ = 0.1;
00025 };
00026
00027 #endif
```

Index