Dungeon crawler 4

Generated by Doxygen 1.8.12

Contents

Chapter 1

Namespace Index

1.1 Namespace List

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

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

Namespace Documentation

4.1 cv Namespace Reference

Functions

- bool isZero (sf::Vector2f v)
- float norm (sf::Vector2f v)
- sf::Vector2f normalized (sf::Vector2f v)
- float distance (sf::Vector2f a, sf::Vector2f b)
- float approxDistance (sf::Vector2f a, sf::Vector2f b)
- float dotP (sf::Vector2f a, sf::Vector2f b)
- std::vector< bool > pathExists (const std::vector< std::vector< bool >> &grid, sf::Vector2i start, std::vector< sf::Vector2i > targets)

Variables

- float **PI** = PI_DEF

4.1.1 Detailed Description

A namespace containing convenience functions for the game. The functionality includes some basic linear algebra functions and other convenient functionality.

4.1.2 Function Documentation

4.1.2.1 approxDistance()

```
float cv::approxDistance (
            sf::Vector2f a,
            sf::Vector2f b )
```

Returns the approximate (square based) distance between two points in the plane.

4.1.2.2 distance()

```
float cv::distance (
            sf::Vector2f a,
            sf::Vector2f b )
```

Returns the distance between two points in the plane.

4.1.2.3 dotP()

```
float cv::dotP (
            sf::Vector2f a,
            sf::Vector2f b )
```

Returns the dot product of two 2D vectors.

4.1.2.4 isZero()

```
bool cv::isZero (
            sf::Vector2f v )
```

Returns whether or not the given vector is zero.

4.1.2.5 norm()

```
float cv::norm (
            sf::Vector2f v )
```

Calculates the norm, or length, of a vector.

4.1.2.6 normalized()

```
sf::Vector2f cv::normalized (
            sf::Vector2f v )
```

Returns a new vector that is the parameter vector v normalized. If the norm of v is zero, returns the zero vector.

4.1.2.7 pathExists()

```
std::vector< bool > cv::pathExists (
            const std::vector< std::vector< bool >> & grid,
            sf::Vector2i start,
            std::vector< sf::Vector2i > targets )
```

Takes as input a grid of penetrable/non-penetrable booleans, start coordinates and a vector of target coordinates, and returns a vector of booleans where each boolean tells whether or not there exists a path between the starting point and the target point at the corresponding index.

Chapter 5

Class Documentation

5.1 s::animation Struct Reference

`#include <settings.hpp>`

Public Member Functions

- void set (sf::Vector2f p, sf::Vector2f dp, float r, float dr, int textureIndex, sf::Vector2f scale, sf::Vector2f origin, float dur)
- void updateTexture (int textureIndex, sf::Vector2f scale, sf::Vector2f origin)
- void restart ()
- void draw (sf::RenderWindow &window, float elapsed)

Public Attributes

- sf::Vector2f **rPos**
- sf::Vector2f **pos**
- sf::Vector2f **dPos**
- float **rRot**
- float **rot**
- float **dRot**
- float **duration**
- float **accumulator**
- bool **active**
- sf::Sprite **sprite**

5.1.1 Detailed Description

Represents a simple translation/rotation animation.

5.1.2 Member Function Documentation

5.1.2.1 draw()

```
void s::animation::draw (
            sf::RenderWindow & window,
            float elapsed )
```

Draws the current state of the animation.

5.1.2.2 restart()

```
void s::animation::restart ( )
```

Restarts the animation.

5.1.2.3 set()

```
void s::animation::set (
            sf::Vector2f p,
            sf::Vector2f dp,
            float r,
            float dr,
            int textureIndex,
            sf::Vector2f scale,
            sf::Vector2f origin,
            float dur )
```

Initializes the animation objects with the given parameters

5.1.2.4 updateTexture()

```
void s::animation::updateTexture (
            int textureIndex,
            sf::Vector2f scale,
            sf::Vector2f origin )
```

Changes the texture of the animation object, as well as the related parameters.

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

- src/settings.hpp
- src/settings.cpp

5.2 Character Class Reference

```
#include <character.hpp>
```

Public Member Functions

- Character (const std::string &n, bool t, float s, sf::Vector2f p, int textureIndex, int shadowIndex, int l=1)
- ~Character ()
- std::string getName () const
- sf::Vector2f getPosition () const
- void setPosition (sf::Vector2f newPosition)
- void move (sf::Vector2f dir, float elapsed)
- sf::Vector2f getHypotheticalPosition (sf::Vector2f dir, float elapsed) const
- int getRotation ()
- void setRotation (int angle)
- Room * getRoom ()
- void setRoom (Room *r)
- void draw (sf::RenderWindow &window, float elapsed)
- void initiateMeleeAttack ()
- std::vector< Item * > & getInventory ()
- void addItem (Item *item)
- bool consumeItem (int i)
- int getHealth () const
- int getMaxHealth () const
- void reducehealth (int damage)
- void sethealth (int newhealth)
- void teleport (sf::Vector2f dpos)
- void givegold (int gold)
- int getgold () const
- void givexp (int amount)
- int getlevel () const
- void setLevel (int newLevel)
- void equipweapon (Weapon *newweapon)
- Weapon * getmeleeweapon () const
- Weapon * getrangedweapon () const

5.2.1 Detailed Description

Represents the character, controlled by the player.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Character()

```
Character::Character (
            const std::string & n,
            bool t,
            float s,
            sf::Vector2f p,
            int textureIndex,
            int shadowIndex,
            int l = 1 )
```

Constructor taking name, type and level with default value 1. This can be used to initiate higher level characters for the purposes of e.g. a save feature.

5.2.2.2 ~Character()

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

The character destructor.

5.2.3 Member Function Documentation

5.2.3.1 addItem()

```
void Character::addItem (
            Item * item )
```

Adds an item to the character's inventory.

5.2.3.2 consumeItem()

```
bool Character::consumeItem (
            int i )
```

Attempts to consume an item. If it could be consumed, returns true, otherwise, returns false.

5.2.3.3 draw()

```
void Character::draw (
            sf::RenderWindow & window,
            float elapsed )
```

Draws the character.

5.2.3.4 equipweapon()

```
void Character::equipweapon (
            Weapon * newweapon )
```

Equips the given weapon.

5.2.3.5 getgold()

```
int Character::getgold ( ) const
```

Returns the character's current amount of gold.

5.2.3.6 getHealth()

```
int Character::getHealth ( ) const
```

Returns the character's current health.

5.2.3.7 getHypotheticalPosition()

```
sf::Vector2f Character::getHypotheticalPosition (
            sf::Vector2f dir,
            float elapsed ) const
```

Returns the position that the character would be in if it moved in direction 'dir' during time 'elapsed', assuming no obstacles.

5.2.3.8 getInventory()

```
std::vector< Item * > & Character::getInventory ( )
```

Retunrs a reference to the character's inventory.

5.2.3.9 getlevel()

```
int Character::getlevel ( ) const
```

Returns the character's current level.

5.2.3.10 getMaxHealth()

```
int Character::getMaxHealth ( ) const
```

Returns the character's current max health.

5.2.3.11 getmeleeweapon()

```
Weapon * Character::getmeleeweapon ( ) const
```

Returns a pointer to the character's current melee weapon.

5.2.3.12 getName()

```
std::string Character::getName ( ) const
```

Returns the name of the character.

5.2.3.13 getPosition()

```
sf::Vector2f Character::getPosition ( ) const
```

Returns the position (in terms of point coordinates in the current room) of the character.

5.2.3.14 getrangedweapon()

```
Weapon * Character::getrangedweapon ( ) const
```

Returns a pointer to the character's current ranged weapon.

5.2.3.15 getRoom()

```
Room * Character::getRoom ( )
```

Returns a pointer to the room currently inhabited by the character.

5.2.3.16 getRotation()

```
int Character::getRotation ( )
```

Returns the character's current rotation, in degrees.

5.2.3.17 givegold()

```
void Character::givegold (
            int gold )
```

Adds the given amount of gold to the character.

5.2.3.18 givexp()

```
void Character::givexp (
            int amount )
```

Adds the given amount of xp to the character and checks for a levelup.

5.2.3.19 initiateMeleeAttack()

```
void Character::initiateMeleeAttack ( )
```

Registers that the character is now initiating a melee attack, for visualisation purposes.

5.2.3.20 move()

```
void Character::move (
            sf::Vector2f dir,
            float elapsed )
```

Moves the character in the given direction, according to the given elapsed time since the last frame.

5.2.3.21 reducehealth()

```
void Character::reducehealth (
            int damage )
```

Reduces the character's health by 'damage', and checks if the character has died.

5.2.3.22 sethealth()

```
void Character::sethealth (
            int newhealth )
```

Sets the character's health to the given new health.

5.2.3.23 setLevel()

```
void Character::setLevel (
            int newLevel )
```

Sets the player's level to the given value and zeros the offset xp.

5.2.3.24 setPosition()

```
void Character::setPosition (
            sf::Vector2f newPosition )
```

Sets the character's position to the given position, without taking any constraints into account. NB! If you wish to move tha character only if possible, use the 'teleport' function instead.

5.2.3.25 setRoom()

```
void Character::setRoom (
            Room * r )
```

Sets the character's room to the given room.

5.2.3.26 setRotation()

```
void Character::setRotation (
            int angle )
```

Sets the character's rotation to 'angle' degrees.

5.2.3.27 teleport()

```
void Character::teleport (
            sf::Vector2f dpos )
```

Teleports the character by the given direction (componentwise), if possible.

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

- src/character.hpp
- src/character.cpp

5.3 Item Class Reference

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

Public Member Functions

- void draw (sf::RenderWindow &window, Character &player)
- std::string getname () const
- int gettype () const
- float getvalue () const
- void dothing (Character &player)
- sf::Vector2f getpos () const
- sf::Sprite & getDropSprite ()
- sf::Sprite & getInventorySprite ()
- Item (std::string namei, int typei, float valuei, int textureIndexi, sf::Vector2f posi, int leveli)
- Item (std::string namei, Weapon *weaponi, int leveli, int textureIndexi, sf::Vector2f posi)
- Item (sf::Vector2f positioni, int leveli)
- ~Item ()
- void applyGoldEffects (Character &player)
- void applyConsumableEffects (Character &player)
- void applyWeaponEffects (Character &player)

5.3.1 Detailed Description

Represents an item, such as a potion, a weapon or a piece of food.

Items can be located in different places, such as dropped on the ground or in someone's inventory. The Item types are defined by an extendable integer variable, as follows:

- 1: Gold, gives player points equal to value variable.
- 2: Healing item, gives player value percentage of max health, up to maximum.
- 3: Weapon. Calls function to give player random weapon (or weapon based on value) when player walks over.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Item() [1/3]

```
Item::Item (
            std::string namei,
            int typei,
            float valuei,
            int textureIndexi,
            sf::Vector2f posi,
            int leveli )
```

Standard Item constructor. Constructs an item of given type based on the specific parameters.

5.3.2.2 Item() [2/3]

```
Item::Item (
            std::string namei,
            Weapon * weaponi,
            int leveli,
            int textureIndexi,
            sf::Vector2f posi )
```

Weapon Item constructor. Constructs a weapon item based on the specific parameters.

5.3.2.3 Item() [3/3]

```
Item::Item (
            sf::Vector2f positioni,
            int leveli )
```

Random Item constructor. Constructs a randomized item.

5.3.2.4 ~Item()

```
Item::~Item ( )
```

Item destructor

5.3.3 Member Function Documentation

5.3.3.1 applyConsumableEffects()

```
void Item::applyConsumableEffects (
            Character & player )
```

Applies the effects of a consumable

5.3.3.2 applyGoldEffects()

```
void Item::applyGoldEffects (
            Character & player )
```

Applies the effects of gold

5.3.3.3 applyWeaponEffects()

```
void Item::applyWeaponEffects (
            Character & player )
```

Applies the effects of a weapon

5.3.3.4 dothing()

```
void Item::dothing (
            Character & player )
```

Performs the action of this item, if it has one.

5.3.3.5 draw()

```
void Item::draw (
            sf::RenderWindow & window,
            Character & player )
```

Draws the item

5.3.3.6 getDropSprite()

```
sf::Sprite & Item::getDropSprite ( )
```

Returns a reference to the item's drop sprite, i.e., the sprite that should be drawn if the item is on the ground.

5.3.3.7 getInventorySprite()

```
sf::Sprite & Item::getInventorySprite ( )
```

Returns a reference to the item's inventory sprite, i.e., the sprite that should be drawn for an inventory.

5.3.3.8 getname()

```
std::string Item::getname ( ) const
```

Returns the name of the item

5.3.3.9 getpos()

```
sf::Vector2f Item::getpos ( ) const
```

Returns the position of this item. If the item is not currently on the ground, the position has no relevance.

5.3.3.10 gettype()

```
int Item::gettype ( ) const
```

Returns the type of the item, represented by an integer as follows: 1 - gold 2 - consumable 3 - weapon other - trinkets or otherwise functionless items

5.3.3.11 getvalue()

```
float Item::getvalue ( ) const
```

Returns the value of the item. The value is a general variable that can be used by different item types as they need.

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

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

5.4 Map Class Reference

Public Member Functions

- **Map** (Character &c)
- Room & **getRoom** ()
- Room & **switchRoom** (int neighbour)

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

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

5.5 MeleeMonster Class Reference

```
#include <monster.hpp>
```

Inheritance diagram for MeleeMonster:

5.6 MeleeWeapon Class Reference

`#include <weapon.hpp>`

Inheritance diagram for MeleeWeapon:

Collaboration diagram for MeleeWeapon:

Public Member Functions

- MeleeWeapon (std::string name, int damage, float minR, float maxR, int txtrIndex)
- MeleeWeapon (int level, float seed)
- ~MeleeWeapon ()
- virtual void attack ()
- virtual float getMinRadius () const override
- virtual float getMaxRadius () const override

Additional Inherited Members

5.6.1 Detailed Description

The MeleeWeapon class represents a close combat weapon, such as a sword or a spear, whose primary attack is direct melee strikes.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 MeleeWeapon() [1/2]

```
MeleeWeapon::MeleeWeapon (
            std::string name,
            int damage,
            float minR,
            float maxR,
            int txtrIndex )
```

The general MeleeWeapon constructor Constructs a specific melee weapon based on the given parameters.

5.6.2.2 MeleeWeapon() [2/2]

```
MeleeWeapon::MeleeWeapon (
            int level,
            float seed )
```

The randomized MeleeWeapon constructor Constructs a random melee weapon based on the player's level.

5.6.2.3 ~MeleeWeapon()

```
MeleeWeapon::~MeleeWeapon ( )
```

The MeleeWeapon destructor

5.6.3 Member Function Documentation

5.6.3.1 attack()

```
void MeleeWeapon::attack ( )  [virtual]
```

Performs an attack.

Reimplemented from Weapon.

5.6.3.2 getMaxRadius()

```
float MeleeWeapon::getMaxRadius ( ) const  [override], [virtual]
```

Returns the maximum radius of the weapon (e.g. the radius of the circle within which the weapon is effective, as long as it is outside the circle defined by minRadius).

Reimplemented from Weapon.

5.6.3.3 getMinRadius()

```
float MeleeWeapon::getMinRadius ( ) const  [override], [virtual]
```

Returns the minimum radius of the weapon (e.g. the radius of the circle within which the weapon is ineffective).

Reimplemented from Weapon.

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

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

5.7 Monster Class Reference

```
#include <monster.hpp>
```

Inheritance diagram for Monster:

Collaboration diagram for Monster:

Public Member Functions

- Monster ()
- virtual ~Monster ()
- std::string getname () const
- int gethealth () const
- int getxponkill () const
- int getattackdamage () const
- float getmovespeed () const
- float getaggrorange () const
- sf::Vector2f getPosition () const
- void setPosition (sf::Vector2f newPos)
- void move (sf::Vector2f dPos)
- bool isactive () const
- void reducehealth (int reducedby)
- bool monsteraggrocheck (const Character &player) const
- float getdistancetoplayer (const Character &player) const
- virtual void monsterattack (Character &player)=0
- virtual void monsterai (Character &player, float elapsed)=0
- void monstermove (sf::Vector2f direction, float elapsed)

Protected Attributes

- std::string **monstername**
- int **health**
- int **xponkill**
- int **attackdamage**
- float **movespeed**
- float **aggrorange**
- sf::Vector2f **position**
- Room * **room**
- bool **aggrostate**
- float **attacktimer**
- float **timebetweenattacks**
- bool **active**
- sf::Sprite * **sprite**
- int **textureIndex**
- sf::Sound **sound**

5.7.1 Detailed Description

An abstract class representing an enemy in the game. A monster is restricted to a room and its purpose is to destroy the player.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Monster()

```
Monster::Monster ( )
```

The monster constructor

5.7.2.2 ~Monster()

```
Monster::~Monster ( )  [virtual]
```

The monster destructor

5.7.3 Member Function Documentation

5.7.3.1 getaggrorange()

```
float Monster::getaggrorange ( ) const
```

Returns the monster's aggression range (the distance at which it can notice the player's presence)

5.7.3.2 getattackdamage()

```
int Monster::getattackdamage ( ) const
```

Returns the monster's attack damage

5.7.3.3 getdistancetoplayer()

```
float Monster::getdistancetoplayer (
            const Character & player ) const
```

Returns the current distance to the player.

5.7.3.4 gethealth()

```
int Monster::gethealth ( ) const
```

Returns the current health of the monster

5.7.3.5 getmovespeed()

```
float Monster::getmovespeed ( ) const
```

Returns the monster's movement speed

5.7.3.6 getname()

```
std::string Monster::getname ( ) const
```

Returns the name of the monster

5.7.3.7 getPosition()

```
sf::Vector2f Monster::getPosition ( ) const
```

Returns the monster's current position

5.7.3.8 getxponkill()

```
int Monster::getxponkill ( ) const
```

Returns the amount of xp the monster gives upon death

5.7.3.9 isactive()

```
bool Monster::isactive ( ) const
```

Returns whether or not the monster is currently active

5.7.3.10 monsteraggrocheck()

```
bool Monster::monsteraggrocheck (
            const Character & player ) const
```

Returns whether or not the monster is currently in aggressive mode.

5.7.3.11 monsterai()

```
virtual void Monster::monsterai (
            Character & player,
            float elapsed )  [pure virtual]
```

Calls the monster AI depending on subclass.

Implemented in MeleeMonster, and RangedMonster.

5.7.3.12 monsterattack()

```
virtual void Monster::monsterattack (
            Character & player )  [pure virtual]
```

Calls the monster attack depending on subclass.

Implemented in MeleeMonster, and RangedMonster.

5.7.3.13 monstermove()

```
void Monster::monstermove (
            sf::Vector2f direction,
            float elapsed )
```

Moves the monster in the given direction depending on the elapsed time.

5.7.3.14 move()

```
void Monster::move (
            sf::Vector2f dPos )
```

Move the monster by the given vector

5.7.3.15 reducehealth()

```
void Monster::reducehealth (
            int reducedby )
```

Reduce monster health. To be called by whatever handles the monster getting hit. Also checks if the monster has died and acts accordingly.

5.7.3.16 setPosition()

```
void Monster::setPosition (
            sf::Vector2f newPos )
```

Sets the monster's position

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

- src/monster.hpp
- src/monster.cpp

5.8 Npc Class Reference

Inheritance diagram for Npc:

Public Member Functions

- Npc (int, sf::Vector2f)
- void setPosition (sf::Vector2f)
- void draw (sf::RenderWindow &, Character &)
- void setRotation (int)
- bool isInRange () const
- std::vector< Item * > & getInventory ()
- void removeFromInventory (int i)

Public Attributes

- NpcType type

Protected Attributes

- std::vector< Item * > **inventory**

5.8.1 Constructor & Destructor Documentation

5.8.1.1 Npc()

```
Npc::Npc (
            int textureIndex,
            sf::Vector2f pos )
```

Constructor taking the texture index (int) and position on the map. A NPC has a constant position unless moved using setPosition() or setRotation()

5.8.2 Member Function Documentation

5.8.2.1 draw()

```
void Npc::draw (
            sf::RenderWindow & window,
            Character & player )
```

Draw the NPC on screen

5.8.2.2 getInventory()

```
std::vector< Item * > & Npc::getInventory ( )
```

Returns a reference to the character's inventory.

5.8.2.3 isInRange()

```
bool Npc::isInRange ( ) const
```

Tells if the NPC is in close proximity of the player The proximity is calculated every time draw() is called.

5.8.2.4 removeFromInventory()

```
void Npc::removeFromInventory (
            int i )
```

Remove the item with index i from character's inventory

5.8.2.5 setPosition()

```
void Npc::setPosition (
            sf::Vector2f newPosition )
```

Sets the position of the NPC, if it needs to be moved to a different location than specified in the constrictor

5.8.2.6 setRotation()

```
void Npc::setRotation (
            int angle )
```

Sets the rotation of the NPC

5.8.3 Member Data Documentation

5.8.3.1 type

```
NpcType Npc::type
```

Enumerator for storing the NPC type (e.g. NPC or SHOPKEEPER)

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

- src/npc.hpp
- src/npc.cpp

5.9 Projectile Class Reference

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

Public Member Functions

- bool isfiredbyplayer ()
- bool isActive ()
- void reset (bool shotbyplayer, int damagein, int radiusin, float speedin, int txtrIndex)
- sf::Vector2f getPosition ()
- void setPosition (sf::Vector2f position)
- sf::Vector2f getVelocity ()
- void setDirection (sf::Vector2f direction)
- float getSpeed () const
- void setSpeed (float newSpeed)
- int getdamage () const
- int getradius () const
- void **deactivate** ()
- void draw (sf::RenderWindow &window, float elapsed)
- Projectile (Room *currentRoom, bool shotbyplayer, int damagein, int radiusin, float speed, int txtrIndex)

5.9.1 Detailed Description

A class representing a projectile fired by the player or by an enemy.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Projectile()

```
Projectile::Projectile (
            Room * currentRoom,
            bool shotbyplayer,
            int damagein,
            int radiusin,
            float speed,
            int txtrIndex )
```

Constructor of the projectile. NB! To add a projectile to a room, do not call this constructor explicitly! Use the Room class interface instead.

5.9.3 Member Function Documentation

5.9.3.1 draw()

```
void Projectile::draw (
            sf::RenderWindow & window,
            float elapsed )
```

Performs necessary calculations, such as computing its new position, and draws itself. The method takes as input a reference to the window to which it needs to draw itself and a float defining the time elapsed since the last frame.

5.9.3.2 getdamage()

```
int Projectile::getdamage ( ) const
```

Returns the damage value of this projectile.

5.9.3.3 getPosition()

```
sf::Vector2f Projectile::getPosition ( )
```

Returns the current position of the projectile in worldspace.

5.9.3.4 getradius()

```
int Projectile::getradius ( ) const
```

Returns the nominal attack radius of this projectile.

5.9.3.5 getSpeed()

```
float Projectile::getSpeed ( ) const
```

Returns the nominal speed of the projectile.

5.9.3.6 getVelocity()

```
sf::Vector2f Projectile::getVelocity ( )
```

Returns the current velocity of the projectile.

5.9.3.7 isActive()

```
bool Projectile::isActive ( )
```

Whether or not this projectile is currently active.

5.9.3.8 isfiredbyplayer()

```
bool Projectile::isfiredbyplayer ( )
```

Whether not this projectile was fired by a player.

5.9.3.9 reset()

```
void Projectile::reset (
            bool shotbyplayer,
            int damagein,
            int radiusin,
            float speedin,
            int txtrIndex )
```

Overwrites the projectile with new information for reuse.

5.9.3.10 setDirection()

```
void Projectile::setDirection (
            sf::Vector2f direction )
```

Sets the projectile's velocity direction to the direction given as parameter. The direction is defined as a 2D unit vector, so the given vector will be normalized.

5.9.3.11 setPosition()

```
void Projectile::setPosition (
            sf::Vector2f position )
```

Sets the projectile's position in worldspace to the position given as parameter.

5.9.3.12 setSpeed()

```
void Projectile::setSpeed (
            float newSpeed )
```

Sets the projectile's speed.

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

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

5.10 RangedMonster Class Reference

`#include <monster.hpp>`

Inheritance diagram for RangedMonster:

Collaboration diagram for RangedMonster:

Public Member Functions

- RangedMonster (std::string namei, int healthi, int xponkilli, int attackdamagei, float movespeedi, float aggrorangei, float projectilespeedi, float attackrangei, Room *roomi, float timebetweenattacksi)
- RangedMonster (sf::Vector2f positioni, Room *roomi, int leveli)
- ~RangedMonster ()
- void monsterattack (Character &player)
- void monsterai (Character &player, float elapsed)

Additional Inherited Members

5.10.1 Detailed Description

A subclass to Monster, RangedMonster represents a monster whose primary attack is a ranged one. These monsters will attempt to destroy the player from a distance.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 RangedMonster() [1/2]

```
RangedMonster::RangedMonster (
            std::string namei,
            int healthi,
            int xponkilli,
            int attackdamagei,
            float movespeedi,
            float aggrorangei,
            float projectilespeedi,
            float attackrangei,
            Room * roomi,
            float timebetweenattacksi )
```

RangedMonster general constructor Constructs a specific ranged monster based on the given parameters.

5.10.2.2 RangedMonster() [2/2]

```
RangedMonster::RangedMonster (
            sf::Vector2f positioni,
            Room * roomi,
            int leveli )
```

RangedMonster random constructor Creates a random ranged monster.

5.10.2.3 ~RangedMonster()

```
RangedMonster::~RangedMonster ( )
```

The RangedMonster Destructor

5.10.3 Member Function Documentation

5.10.3.1 monsterai()

```
void RangedMonster::monsterai (
            Character & player,
            float elapsed )  [virtual]
```

Tries to find the player and advance towards them.

Implements Monster.

5.10.3.2 monsterattack()

```
void RangedMonster::monsterattack (
            Character & player )  [virtual]
```

Fires at the player if in range.

Implements Monster.

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

- src/monster.hpp
- src/monster.cpp

5.11 RangedWeapon Class Reference

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

Inheritance diagram for RangedWeapon:

Collaboration diagram for RangedWeapon:

Public Member Functions

- RangedWeapon (std::string name, int damage, int txtrIndex)
- RangedWeapon (int level, float seed)
- ~RangedWeapon ()
- virtual void attack ()

Additional Inherited Members

5.11.1 Detailed Description

The RangedWeapon class represents a ranged weapon, such as a bow or a magic staff, whose primary attack is launching projectiles.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 RangedWeapon() [1/2]

```
RangedWeapon::RangedWeapon (
            std::string name,
            int damage,
            int txtrIndex )
```

The general RangedWeapon constructor Constructs a specific ranged weapon based on the given parameters.

5.11.2.2 RangedWeapon() [2/2]

```
RangedWeapon::RangedWeapon (
            int level,
            float seed )
```

The randomized RangedWeapon constructor Constructs a random ranged weapon based on the player's level.

5.11.2.3 ~RangedWeapon()

```
RangedWeapon::~RangedWeapon ( )
```

The RangedWeapon destructor

5.11.3 Member Function Documentation

5.11.3.1 attack()

```
void RangedWeapon::attack ( )  [virtual]
```

Performs an attack.

Reimplemented from Weapon.

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

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

5.12 Room Class Reference

Public Member Functions

- Room (std::string const file, Character ∗character)
- Room (int width, int height, float p, int randomGenIterations, std::vector< bool > entrances, Character ∗character)
- ∼Room ()
- int getWidth () const
- int getHeight () const
- bool hasCoordinate (int x, int y)
- bool hasPosition (sf::Vector2f pos)
- sf::Vector2i getOffsetDirection (sf::Vector2f pos)
- Tile & getTile (int x, int y)
- Tile & getTile (sf::Vector2f pos)
- std::vector< sf::Vector2i > getNeighbours (int x, int y, bool includingSelf, bool includingDiagonals, bool includingOutsiders)
- void performAttack (bool friendly, sf::Vector2f source, sf::Vector2f direction, const Weapon &weapon)
- void draw (sf::RenderWindow &window)
- void drawProjectiles (sf::RenderWindow &window, float elapsed)
- void drawmonsters (float elapsed)
- void drawnpcs (sf::RenderWindow &window)
- std::vector< Npc ∗ > & getNpcs ()
- void drawitems (sf::RenderWindow &window)
- void additem (Item ∗newitem)
- void checkDrops ()
- void print ()
- std::vector< std::vector< bool > > getPenetrabilityMap ()
- sf::Sprite ∗ getSprite ()
- void deactivateSprite (sf::Sprite ∗sprite)
- Projectile & createProjectile (bool shotbyplayer, int damagein, int radiusin, float speedin, int txtrIndex)
- std::vector< Monster ∗ > & getmonsters ()
- std::vector< Item ∗ > & getitems ()
- Character ∗ getcharacter ()
- void addmonster (Monster ∗monsteri)
- void addNpc (Npc ∗npc)

5.12.1 Constructor & Destructor Documentation

5.12.1.1 Room() [1/2]

```
Room::Room (
            std::string const file,
            Character * character )
```

Constructor creating the room by loading it from the given file.

5.12.1.2 Room() [2/2]

```
Room::Room (
            int width,
            int height,
            float p,
            int randomGenIterations,
            std::vector< bool > entrances,
            Character * character )
```

Constructor creating a random generated room based on the given parameters.

5.12.1.3 ~Room()

```
Room::~Room ( )
```

The Room destructor

5.12.2 Member Function Documentation

5.12.2.1 additem()

```
void Room::additem (
            Item * newitem )
```

Adds an item to the ground.

5.12.2.2 addmonster()

```
void Room::addmonster (
            Monster * monsteri )
```

Adds the given monster to the room.

5.12.2.3 addNpc()

```
void Room::addNpc (
            Npc * npc )
```

Adds the given NPC to the room.

5.12.2.4 checkDrops()

```
void Room::checkDrops ( )
```

Identifies if there is any item on the ground within reach from the the player. If one such item is found, it is picked up. Therefore, a maximum of 1 item can be picked up each frame.

5.12.2.5 createProjectile()

```
Projectile & Room::createProjectile (
            bool shotbyplayer,
            int damagein,
            int radiusin,
            float speedin,
            int txtrIndex )
```

Adds a projectile with the input parameters as constructor parameters to this room and returns the projectile index of this projectile in the Room's projectile buffer. NB! Always use this method to add a projectile - never use the Projectile class' explicit constructor! (This method will try to reactivate a deactivated projectile class in its projectile buffer and assign these parameters to it.)

5.12.2.6 deactivateSprite()

```
void Room::deactivateSprite (
            sf::Sprite * sprite )
```

Deactivates a sprite for other instances to use.

5.12.2.7 draw()

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

Draws the room.

5.12.2.8 drawitems()

```
void Room::drawitems (
            sf::RenderWindow & window )
```

Draws the items on the ground.

5.12.2.9 drawmonsters()

```
void Room::drawmonsters (
            float elapsed )
```

Calls the monster AI for each monster in this room and draws them.

5.12.2.10 drawnpcs()

```
void Room::drawnpcs (
            sf::RenderWindow & window )
```

Calls any necessary frame calls for the NPCs in this room and draws them.

5.12.2.11 drawProjectiles()

```
void Room::drawProjectiles (
            sf::RenderWindow & window,
            float elapsed )
```

Updates the projectiles in the room and draws them.

5.12.2.12 getcharacter()

```
Character * Room::getcharacter ( )
```

Returns a pointer to the room's character.

5.12.2.13 getHeight()

```
int Room::getHeight ( ) const
```

Returns the height of the room (in blocks)

5.12.2.14 getitems()

```
std::vector< Item * > & Room::getitems ( )
```

Returns a reference to the tile vector of the room, i.e. pointers to all the items that are currently on the ground in the room.

5.12.2.15 getmonsters()

```
std::vector< Monster * > & Room::getmonsters ( )
```

Returns a reference to the monster vector of the room, i.e., pointers to all the monsters that are currently in the room (active or inactive).

5.12.2.16 getNeighbours()

```
std::vector< sf::Vector2i > Room::getNeighbours (
            int x,
            int y,
            bool includingSelf,
            bool includingDiagonals,
            bool includingOutsiders )
```

Returns a vector of neighbour coordinates to teh coordinate at (x, y). If 'includingSelf' is true, the original coordinate is included. If 'includingDiagonals' is true, the diagonal neighbours are included. If 'includingOutsiders' is true, such neighbours that are out of bounds are included.

5.12.2.17 getNpcs()

```
std::vector< Npc * > & Room::getNpcs ( )
```

Returns a reference to the vector containing pointers to all the NPCs in this room.

5.12.2.18 getOffsetDirection()

```
sf::Vector2i Room::getOffsetDirection (
            sf::Vector2f pos )
```

Returns a 2D vector describing in which direction(s) the given position is out of bounds. If the given position is still within bounds, the returned vector is the zero vector.

5.12.2.19 getPenetrabilityMap()

```
std::vector< std::vector< bool > > Room::getPenetrabilityMap ( )
```

Returns a 2D vector of booleans representing the room, where a 'true' value represents a non-penetrable cell and a 'false' value represents a penetrable cell.

5.12.2.20 getSprite()

```
sf::Sprite * Room::getSprite ( )
```

Returns refernce to a sprite for the sprite resuse ecosystem.

5.12.2.21 getTile() [1/2]

```
Tile & Room::getTile (
            int x,
            int y )
```

Returns a reference to the tile at coordinates (x, y). If (x, y) are invalid coordinates, an error is thrown.

5.12.2.22 getTile() [2/2]

```
Tile & Room::getTile (
            sf::Vector2f pos )
```

Returns a reference to the tile at position pos. If pos is out of bounds, an error is thrown.

5.12.2.23 getWidth()

```
int Room::getWidth ( ) const
```

Returns the width of the room (in blocks)

5.12.2.24 hasCoordinate()

```
bool Room::hasCoordinate (
            int x,
            int y )
```

Checks whether the room contains the coordinates at (x, y) (= not out of bounds).

5.12.2.25 hasPosition()

```
bool Room::hasPosition (
            sf::Vector2f pos )
```

Checks whether the room contains the given position (= not out of bounds).

5.12.2.26 performAttack()

```
void Room::performAttack (
            bool friendly,
            sf::Vector2f source,
            sf::Vector2f direction,
            const Weapon & weapon )
```

Performs an attack from the given position, in the given direction, with the given weapon. If friendly, the attack targets only monsters. If not, it targets only monsters.

5.12.2.27 print()

```
void Room::print ( )
```

Prints the room to std::cout.

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

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

5.13 roomContainer Struct Reference

Public Attributes

- RoomType **type**
- bool **active**
- std::string **roomPath**
- int **neighbourEast**
- int **neighbourSouth**
- int **neighbourWest**
- int **neighbourNorth**

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

- src/map.hpp

5.14 Shopkeeper Class Reference

Inheritance diagram for Shopkeeper:

Collaboration diagram for Shopkeeper:

Public Member Functions

- **Shopkeeper** (int, sf::Vector2f)

Additional Inherited Members

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

- src/npc.hpp
- src/npc.cpp

5.15 Tile Class Reference

```
#include <tile.hpp>
```

Public Member Functions

- Tile (int type, sf::Vector2f position, sf::Vector2i index, sf::Sprite *freeSprite)
- bool isPenetrable () const
- std::string toString () const

5.15.1 Detailed Description

The Tile class represents a cell in a room's grid. A tile holds information about its texture, position and whether or not it it can be walked on.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Tile()

```
Tile::Tile (
            int type,
            sf::Vector2f position,
            sf::Vector2i index,
            sf::Sprite * freeSprite )
```

Tile constructor Constructs a tile of the given type, at the given index in a Room grid and on the given position, using the the given sprite.

5.15.3 Member Function Documentation

5.15.3.1 isPenetrable()

```
bool Tile::isPenetrable ( ) const
```

Returns whether or not entities can go through this tile.

5.15.3.2 toString()

```
std::string Tile::toString ( ) const
```

Returns a string representation of the tile.

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

- src/tile.hpp
- src/tile.cpp

5.16 Weapon Class Reference

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

Inheritance diagram for Weapon:

Public Member Functions

- Weapon (std::string n, int t, int d, int txtrIndex)
- virtual ~Weapon ()
- std::string getName () const
- int getType () const
- int getDamage () const
- virtual float getMinRadius () const
- virtual float getMaxRadius () const
- int getProjectilespeed () const
- Projectile & createProjectile (Room &room)
- float getcooldown () const
- int gettextureindex () const
- virtual void attack ()

Protected Member Functions

- Weapon (int level, float seed)

Protected Attributes

- std::string **name**
- int **type**
- int **damage**
- int **textureIndex**
- float **projectilespeed**
- float **cooldown**
- int **levels**
- float **seeds**

5.16.1 Detailed Description

The general Weapon class, representing a weapon in the game. This class can have several child classes for polymorphic behaviour. Additionally, the main weapon type is defined by an integer as follows:

- 1: Melee
- 2: Ranged

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Weapon() [1/2]

```
Weapon::Weapon (
            std::string n,
            int t,
            int d,
            int txtrIndex )
```

The general Weapon constructor Constructs a specific weapon based on the given parameters.

5.16.2.2 ~Weapon()

```
Weapon::~Weapon ( )  [virtual]
```

The Weapon destructor.

5.16.2.3 Weapon() [2/2]

```
Weapon::Weapon (
            int level,
            float seed )  [protected]
```

A partial constructor to be called by child constructors.

5.16.3 Member Function Documentation

5.16.3.1 attack()

```
void Weapon::attack ( )  [virtual]
```

Performs an attack.

Reimplemented in MeleeWeapon, and RangedWeapon.

5.16.3.2 createProjectile()

```
Projectile & Weapon::createProjectile (
            Room & room )
```

Creates a projectile in the given room as fired by this weapon, and returns a reference to it.

5.16.3.3 getcooldown()

```
float Weapon::getcooldown ( ) const
```

Returns the cooldown of this weapon.

5.16.3.4 getDamage()

```
int Weapon::getDamage ( ) const
```

Returns the damage of the weapon

5.16.3.5 getMaxRadius()

```
float Weapon::getMaxRadius ( ) const  [virtual]
```

Returns the maximum radius of the weapon (e.g. the radius of the circle within which the weapon is effective, as long as it is outside the circle defined by minRadius), if it is relevant. If it is not, e.g. for range-only weapons, returns a negative number.

Reimplemented in MeleeWeapon.

5.16.3.6 getMinRadius()

```
float Weapon::getMinRadius ( ) const  [virtual]
```

Returns the minimum radius of the weapon (e.g. the radius of the circle within which the weapon is ineffective), if it is relevant. If it is not, e.g. for range-only weapons, returns a negative number.

Reimplemented in MeleeWeapon.

5.16.3.7 getName()

```
std::string Weapon::getName ( ) const
```

Returns the name of the weapon

5.16.3.8 getProjectilespeed()

```
int Weapon::getProjectilespeed ( ) const
```

Returns the projectile speed of projectiles fired by this weapon.

5.16.3.9 gettextureindex()

```
int Weapon::gettextureindex ( ) const
```

Returns the texture index of this weapon.

5.16.3.10 getType()

```
int Weapon::getType ( ) const
```

Returns the type of the weapon, as an integer as follows:

- 1: Melee
- 2: Ranged

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

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

Index