Dungeon crawler

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

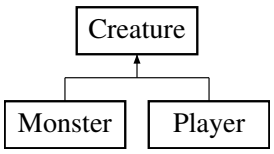
Class Documentation

4.1 Creature Class Reference

Base class for every "alive" entity in the dungeon.

`#include <Creature.hpp>`

Inheritance diagram for Creature:



Public Member Functions

- Creature (const std::string &type, const std::string &name, int maxHealth, float maxVelocity, const sf::↩ Vector2< float > &initialPos, sf::RenderWindow &window, const Room &room, int base_damage, sf::Texture &t, std::ostream &logger=std::cout, const Inventory &inventory=Inventory())

  The constructor.
- bool IsAlive () const

  Checks if the creature is alive.
- const std::string & GetDescription () const
- float TakeHit (float base_damage, const Creature &c2)

  Handles the logic when this creature is attacked.
- float Attack (Creature &c2, const Sword &sword={}) const

  Attacks another creature with a given item.
- virtual void **Update** ()

  Updates state of the creature. Later probably need to pass something cleverer, like game class, tho i am not sure yet.
- void Draw ()

  Draws the creature's sprite in a given window.
- void SetVelocity (const sf::Vector2< float > &newVelocity)

  Setter for velocity.
- void SetVelocityX (float nvx)

  Sets x velocity.

- void SetVelocityY (float nvy)

  Sets y velocity.
- void SetTexture (const sf::Texture &t)

  Sets texture.
- virtual void SetPosition (const sf::Vector2< float > &position)

  Sets position.
- virtual void SetRoom (Room &room)

  Sets room; will be overriden by player so that changing rooms spawns monsters.
- void DrawHealthBar (sf::RenderWindow &window)

  draws the healthbar of the creature
- const sf::Vector2< float > & GetPosition () const
- const Room & GetRoom () const
- Inventory & GetInventory ()

Protected Member Functions

- void TakeDamage_ (float damage)

  Reduces health by damage. If damage is greater then health, sets health to 0.
- virtual std::vector< Room > GetAvailableRooms ()

  Returns the rooms the creature can go too. By default returns only current room; however, if the creature is a player and the monsters are cleared the next room is also available.
- void **UpdatePosition** ()

  Updates position based on current position, velocity and available rooms. If position + velocity is in one of the available rooms then the position is updated, if not then it stays the same.
- void **UpdateRotation** ()

  Updated rotation so that the creatures faces the direction it is supposed to face, as specified by GetFacingDirection() .
- virtual sf::Vector2f GetFacingDirection ()

  Returns the facing direction of the creature; the default case is just face the direction of moving; however, the Player class overrides the function and Player objects face the cursor.
- void TurnToDirection (float dx, float dy)

  Turns the creature to the direction of the vector (dx, dy)

Protected Attributes

- const float **maxVelocity_**

  max movement speed of the creature\nNote: this is the limit for vx and vy separately.
- const float **maxHealth_**

  max health of the creature
- float **health_**

  current health of the creature.
- int **base_damage_**

  damage that creature deals with no items
- const std::string **type_**

  type of the character (e.g. monster)
- std::string **name_**

  name of the character
- std::string **description_**

  description in the format "Creature of type <type_> named <name>"
- Inventory **inventory_**

the inventory_ of the creature

- sf::Texture & **texture_**

  texture
- sf::Vector2< float > **position_**

  position of the creature on the screen
- sf::Vector2< float > **velocity_**

  current velocity of the creature
- sf::RenderWindow & **window_**

  window in which the creature is supposed to be drawn
- std::ostream & **logger_**

  stream to log information about the class.
- Room **room_**

  room the creature is in.
- sf::RectangleShape **creatureRect_**

  The rectangle representing the creature.

4.1.1 Detailed Description

Base class for every "alive" entity in the dungeon.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Creature()

```
Creature::Creature (
            const std::string & type,
            const std::string & name,
            int maxHealth,
            float maxVelocity,
            const sf::Vector2< float > & initialPos,
            sf::RenderWindow & window,
            const Room & room,
            int base_damage,
            sf::Texture & t,
            std::ostream & logger = std::cout,
            const Inventory & inventory = Inventory() )
```

The constructor.

Parameters

type	type of the creature
name	name of the creature
maxHealth	maximum health of the creature
maxVelocity	maximum velocity of the creature
initialPos	initial position of the creature
window	window the creature is going to be drawn
room	room the creature is initially in
base_damage	the damage that creature does without any items
t	texture of the creature
logger	the stream to log info to
inventory	inventory of the creature

4.1.3 Member Function Documentation

4.1.3.1 Attack()

```
float Creature::Attack (
            Creature & c2,
            const Sword & sword = {} ) const
```

Attacks another creature with a given item.

Parameters

c2	The creature to attack
sword	The item to attack with; if sword = {}, no buffs are applied

Returns

damage dealt

4.1.3.2 Draw()

```
void Creature::Draw ( )
```

Draws the creature's sprite in a given window.

Parameters

camera	current camera state

4.1.3.3 DrawHealthBar()

```
void Creature::DrawHealthBar (
            sf::RenderWindow & window )
```

draws the healthbar of the creature

Parameters

window	the window where the healthbar is drawn to

4.1.3.4 GetAvailableRooms()

```
std::vector< Room > Creature::GetAvailableRooms ( )  [protected], [virtual]
```

Returns the rooms the creature can go too. By default returns only current room; however, if the creature is a player and the monsters are cleared the next room is also available.

Returns

4.1.3.5 GetDescription()

```
const std::string & Creature::GetDescription ( ) const
```

Returns

description_

4.1.3.6 GetFacingDirection()

```
sf::Vector2f Creature::GetFacingDirection ( )  [protected], [virtual]
```

Returns the facing direction of the creature; the default case is just face the direction of moving; however, the Player class overrides the function and Player objects face the cursor.

Returns

The vector specifying the direction the creature should face.

4.1.3.7 GetInventory()

```
Inventory & Creature::GetInventory ( )
```

Returns

inventory_ of the creture

4.1.3.8 GetPosition()

```
const sf::Vector2< float > & Creature::GetPosition ( ) const
```

Returns

position

4.1.3.9 GetRoom()

```
const Room & Creature::GetRoom ( ) const
```

Returns

current room

4.1.3.10 IsAlive()

```
bool Creature::IsAlive ( ) const  [inline]
```

Checks if the creature is alive.

Returns

true if the creature is alive, false otherwise

4.1.3.11 SetPosition()

```
void Creature::SetPosition (
            const sf::Vector2< float > & position )  [virtual]
```

Sets position.

Parameters

position	new position

4.1.3.12 SetRoom()

```
void Creature::SetRoom (
            Room & room )  [virtual]
```

Sets room; will be overriden by player so that changing rooms spawns monsters.

Parameters

room	new room

Reimplemented in Player.

4.1.3.13 SetTexture()

```
void Creature::SetTexture (
            const sf::Texture & t )
```

Sets texture.

Parameters

t	new texture

4.1.3.14 SetVelocity()

```
void Creature::SetVelocity (
            const sf::Vector2< float > & newVelocity )
```

Setter for velocity.

Parameters

newVelocity	

4.1.3.15 SetVelocityX()

```
void Creature::SetVelocityX (
            float nvx )
```

Sets x velocity.

Parameters

nvx	new x velocity.

4.1.3.16 SetVelocityY()

```
void Creature::SetVelocityY (
            float nvy )
```

Sets y velocity.

Parameters

nvy	new y velocity.

4.1.3.17 TakeDamage_()

```
void Creature::TakeDamage_ (
            float damage )  [protected]
```

Reduces health by damage. If damage is greater then health, sets health to 0.

Parameters

damage	amount to reduce health_ by

4.1.3.18 TakeHit()

```
float Creature::TakeHit (
            float base_damage,
            const Creature & c2 )
```

Handles the logic when this creature is attacked.

Parameters

base_damage	Base attack damage (actual damage may be later recalculated somehow)
c2	the creature who attacks

Returns

the damage taken

4.1.3.19 TurnToDirection()

```
void Creature::TurnToDirection (
            float dx,
            float dy )  [protected]
```

Turns the creature to the direction of the vector (dx, dy)

Parameters

dx	x coordinate of the turn vector
dy	y coordinate of the turn vector

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

- Creature/Creature.hpp
- Creature/Creature.cpp

4.2 Dungeon Class Reference

Class representing a dungeon_.

```
#include <dungeon.hpp>
```

Public Member Functions

- **Dungeon** ()

  Constructor for the Dungeon class.
- void GenerateDungeon (std::vector< Room > &rooms, std::vector< Room > &corridors, int numRooms, int TILE_SIZE, int WINDOW_WIDTH, int WINDOW_HEIGHT)

  Generates a dungeon_ with rooms_.

4.2.1 Detailed Description

Class representing a dungeon_.

4.2.2 Member Function Documentation

4.2.2.1 GenerateDungeon()

```
void Dungeon::GenerateDungeon (
            std::vector< Room > & rooms,
            std::vector< Room > & corridors,
            int numRooms,
            int TILE_SIZE,
            int WINDOW_WIDTH,
            int WINDOW_HEIGHT )
```

Generates a dungeon_ with rooms_.

Parameters

rooms	Vector to store generated rooms_.
corridors	Vector to store generated corridors_.
numRooms	Number of rooms_ to generate.
TILE_SIZE	Size of a tile.
WINDOW_WIDTH	Width of the game window.
WINDOW_HEIGHT	Height of the game window.

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

- dungeon.hpp
- dungeon.cpp

4.3 Game Class Reference

The Game class manages the main game loop and logic.

```
#include <game.hpp>
```

Public Member Functions

- **Game** ()
  Constructor.
- ~**Game** ()
  Destructor.
- void **Run** ()
  Start the game loop.

4.3.1 Detailed Description

The Game class manages the main game loop and logic.

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

- game.hpp
- game.cpp

4.4 GameOverScreen Class Reference

Class for managing the game over screen.

`#include <gameOverScreen.hpp>`

Public Member Functions

- void Render (sf::RenderWindow &window, sf::Font &font, std::string text, sf::Color textColor, sf::Color buttonColor)

  Renders the game over screen.
- sf::RectangleShape GetPlayAgainButton ()

  Gets the play again button shape.

4.4.1 Detailed Description

Class for managing the game over screen.

4.4.2 Member Function Documentation

4.4.2.1 GetPlayAgainButton()

```
sf::RectangleShape GameOverScreen::GetPlayAgainButton ( )
```

Gets the play again button shape.

Returns

The play again button as an SFML RectangleShape.

4.4.2.2 Render()

```
void GameOverScreen::Render (
            sf::RenderWindow & window,
            sf::Font & font,
            std::string text,
            sf::Color textColor,
            sf::Color buttonColor )
```

Renders the game over screen.

Parameters

window	The render window.
font	The font to use for text rendering.
text	The main text to display.
textColor	The color of the main text.
buttonColor	The color of the play again button.

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

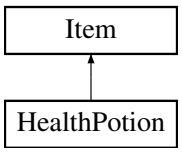
- gameOverScreen.hpp
- gameOverScreen.cpp

4.5 HealthPotion Class Reference

A health potion - an item that heals the player by the given amount.

`#include <Item.hpp>`

Inheritance diagram for HealthPotion:



Public Member Functions

- **HealthPotion** (float hpRestored)
- float **GetHpRestored** () const

Public Member Functions inherited from Item

- **Item** (const std::string &name, bool isSword)
- const std::string & **GetName** () const
- bool **IsSword** () const

Additional Inherited Members

Static Public Member Functions inherited from Item

- static void Draw (sf::RenderWindow &window, const sf::Vector2f &pos, float maxSize, const sf::Texture &t)

  Draws the item.

Protected Attributes inherited from Item

- std::string **name_**

  name of the item.
- bool **isSword_** {}

  true if the item is a sword, false otherwise.

4.5.1 Detailed Description

A health potion - an item that heals the player by the given amount.

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

- Item.hpp

4.6 Inventory Class Reference

Class defining an inventory_ to manage items.
All the items are stored in a vector; each item is either sword or a health potion. The item in use is highlighted while drawing the inventory. Additionally there are two vectors swords_ and potions_ which store swords and potions separately (this is done to return the object of type sword/potion)

`#include <inventory.hpp>`

Public Member Functions

- void AddSword (const Sword &item, int quantity)

  Adds a sword to the inventory.
- void AddPotion (const HealthPotion &item, int quantity)

  Adds a potion to the inventory.
- int GetSize ()

  Get the current size of the inventory_.
- void Draw (sf::RenderWindow &window, int itemInUse)

  Draws the inventory.
- Sword GetSword (int index)
- HealthPotion GetPotion (int index)
- int GetHealingAmount (int index)
- bool IsSword (int i)

4.6.1 Detailed Description

Class defining an inventory_ to manage items.
All the items are stored in a vector; each item is either sword or a health potion. The item in use is highlighted while drawing the inventory. Additionally there are two vectors swords_ and potions_ which store swords and potions separately (this is done to return the object of type sword/potion)

4.6.2 Member Function Documentation

4.6.2.1 AddPotion()

```
void Inventory::AddPotion (
            const HealthPotion & item,
            int quantity )  [inline]
```

Adds a potion to the inventory.

Parameters

item	potion to add
quantity	the number of potions to add

4.6.2.2 AddSword()

```
void Inventory::AddSword (
            const Sword & item,
            int quantity )  [inline]
```

Adds a sword to the inventory.

Parameters

item	sword to add
quantity	the number of swords to add

4.6.2.3 Draw()

```
void Inventory::Draw (
            sf::RenderWindow & window,
            int itemInUse )
```

Draws the inventory.

Parameters

window	window to draw the inventory in
itemInUse	item that is currently in use (for highlighting)

4.6.2.4 GetHealingAmount()

```
int Inventory::GetHealingAmount (
            int index )
```

Parameters

index	position of the potion

Returns

The healing from the potion in the given position.

4.6.2.5 GetPotion()

```
HealthPotion Inventory::GetPotion (
            int index )
```

Parameters

index	position of the potion

Returns

The potion in the given position.

4.6.2.6 GetSize()

```
int Inventory::GetSize ( )
```

Get the current size of the inventory_.

Returns

The size of the inventory_.

4.6.2.7 GetSword()

```
Sword Inventory::GetSword (
            int index )
```

Parameters

index	position of the sword

Returns

The sword in the given position.

4.6.2.8 IsSword()

```
bool Inventory::IsSword (
            int i )  [inline]
```

Parameters

i	the position to check

Returns

True if the item in the position i is a sword; false otherwise.

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

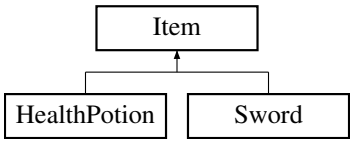
- inventory.hpp
- inventory.cpp

4.7 Item Class Reference

Basic class for items in the inventory.

`#include <Item.hpp>`

Inheritance diagram for Item:



Public Member Functions

- **Item** (const std::string &name, bool isSword)
- const std::string & **GetName** () const
- bool **IsSword** () const

Static Public Member Functions

- static void Draw (sf::RenderWindow &window, const sf::Vector2f &pos, float maxSize, const sf::Texture &t)

  Draws the item.

Protected Attributes

- std::string **name_**

  name of the item.
- bool **isSword_** {}

  true if the item is a sword, false otherwise.

4.7.1 Detailed Description

Basic class for items in the inventory.

4.7.2 Member Function Documentation

4.7.2.1 Draw()

```
void Item::Draw (
            sf::RenderWindow & window,
            const sf::Vector2f & pos,
            float maxSize,
            const sf::Texture & t )  [static]
```

Draws the item.

Parameters

window	Window to draw the item in.
pos	Position to draw the item at.
maxSize	Maximum of allowed width and height.
t	Texture to use.

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

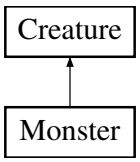
- Item.hpp
- item.cpp

4.8 Monster Class Reference

Class for enemies.

`#include <GameCharacter.hpp>`

Inheritance diagram for Monster:



Public Member Functions

- **Monster** (const std::string &type, const std::string &name, int max_health, float max_velocity, const sf::↩ Vector2< float > &initial_pos, sf::RenderWindow &window, const Room &room, int base_damage, sf::Texture &texture, std::ostream &logger=std::cout, Inventory inventory=Inventory())
- void tick (Player &p)

  Updates the monster speed every fixed amount of seconds; monsters usually try to approach the player but can go other direction with small chance.

Public Member Functions inherited from Creature

- Creature (const std::string &type, const std::string &name, int maxHealth, float maxVelocity, const sf::↩ Vector2< float > &initialPos, sf::RenderWindow &window, const Room &room, int base_damage, sf::Texture &t, std::ostream &logger=std::cout, const Inventory &inventory=Inventory())

  The constructor.
- bool IsAlive () const

  Checks if the creature is alive.
- const std::string & GetDescription () const
- float TakeHit (float base_damage, const Creature &c2)

  Handles the logic when this creature is attacked.
- float Attack (Creature &c2, const Sword &sword={}) const

  Attacks another creature with a given item.
- virtual void **Update** ()

Updates state of the creature. Later probably need to pass something cleverer, like game class, tho i am not sure yet.

- void Draw ()

  Draws the creature's sprite in a given window.
- void SetVelocity (const sf::Vector2< float > &newVelocity)

  Setter for velocity.
- void SetVelocityX (float nvx)

  Sets x velocity.
- void SetVelocityY (float nvy)

  Sets y velocity.
- void SetTexture (const sf::Texture &t)

  Sets texture.
- virtual void SetPosition (const sf::Vector2< float > &position)

  Sets position.
- virtual void SetRoom (Room &room)

  Sets room; will be overriden by player so that changing rooms spawns monsters.
- void DrawHealthBar (sf::RenderWindow &window)

  draws the healthbar of the creature
- const sf::Vector2< float > & GetPosition () const
- const Room & GetRoom () const
- Inventory & GetInventory ()

Additional Inherited Members

Protected Member Functions inherited from Creature

- void TakeDamage_ (float damage)

  Reduces health by damage. If damage is greater then health, sets health to 0.
- virtual std::vector< Room > GetAvailableRooms ()

  Returns the rooms the creature can go too. By default returns only current room; however, if the creature is a player and the monsters are cleared the next room is also available.
- void **UpdatePosition** ()

  Updates position based on current position, velocity and available rooms. If position + velocity is in one of the available rooms then the position is updated, if not then it stays the same.
- void **UpdateRotation** ()

  Updated rotation so that the creatures faces the direction it is supposed to face, as specified by GetFacingDirection() .
- virtual sf::Vector2f GetFacingDirection ()

  Returns the facing direction of the creature; the default case is just face the direction of moving; however, the Player class overrides the function and Player objects face the cursor.
- void TurnToDirection (float dx, float dy)

  Turns the creature to the direction of the vector (dx, dy)

Protected Attributes inherited from Creature

- const float **maxVelocity_**

  max movement speed of the creature\nNote: this is the limit for vx and vy separately.
- const float **maxHealth_**

  max health of the creature
- float **health_**

  current health of the creature.

- int **base_damage_**

  damage that creature deals with no items
- const std::string **type_**

  type of the character (e.g. monster)
- std::string **name_**

  name of the character
- std::string **description_**

  description in the format "Creature of type <type_> named <name>"
- Inventory **inventory_**

  the inventory_ of the creature
- sf::Texture & **texture_**

  texture
- sf::Vector2< float > **position_**

  position of the creature on the screen
- sf::Vector2< float > **velocity_**

  current velocity of the creature
- sf::RenderWindow & **window_**

  window in which the creature is supposed to be drawn
- std::ostream & **logger_**

  stream to log information about the class.
- Room **room_**

  room the creature is in.
- sf::RectangleShape **creatureRect_**

  The rectangle representing the creature.

4.8.1 Detailed Description

Class for enemies.

4.8.2 Member Function Documentation

4.8.2.1 tick()

```
void Monster::tick (
            Player & p )
```

Updates the monster speed every fixed amount of seconds; monsters usually try to approach the player but can go other direction with small chance.

Parameters

p	the player to approach

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

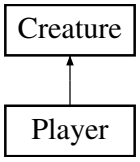
- Creature/GameCharacter.hpp
- Creature/GameCharacter.cpp

4.9 Player Class Reference

Class for the Player.

`#include <GameCharacter.hpp>`

Inheritance diagram for Player:



Public Member Functions

- **Player** (const std::string &type, const std::string &name, int maxHealth, float maxVelocity, const sf::Vector2< float > &initialPos, sf::RenderWindow &window, Room room, sf::Texture &texture, std::ostream &logger=std::cout, Inventory inventory=Inventory())
- int GetRoomIndex () const
- sf::Vector2f GetPosition ()
- void SetRoom (Room &room) override

  Sets the room.
- void SetRoom (Room &room, std::vector< Monster * > &monsters, std::vector< sf::Vector2f > &potionPos)

  Sets the room, if needed spawns the monster and the potions.
- void Update (std::vector< Monster * > &monsters, std::vector< sf::Vector2f > potions)

  Updates the player; if needed spawns the monsters and potions.
- void SetRooms (const std::vector< Room > &rooms)

  Sets rooms.
- void SetMonstersCleared (bool monstersCleared)

  Set monstersCleared param.
- void SetItemInUse (int index)

  Sets itemInUse.
- int GetItemInUse () const
- void **TryHealing** ()

  Tries to heal the player; if the current item in use is not a health potion or there are no health potions left does nothing; otherwise heals the player by the amount specified by the potion.
- void TryAttack (const std::vector< Monster * > &monsters)

  Tries to attack; if there are no monsters close enough, does nothing; otherwise attacks the closest one.
- void TryPickup (std::vector< sf::Vector2f > &potionPositions)

  Tries to pick up a potion; if there are no potions close enough, does nothing; otherwise picks up the closest one.
- void SpawnMonsters (std::vector< Monster * > &res)

  Spawns the monsters in the room with the player. If the room is a corridor does nothing.
- void SpawnPotion (std::vector< sf::Vector2f > &pos)

  Spawns potions in the room with the player with predefined probability.

Public Member Functions inherited from Creature

- Creature (const std::string &type, const std::string &name, int maxHealth, float maxVelocity, const sf::↩ Vector2< float > &initialPos, sf::RenderWindow &window, const Room &room, int base_damage, sf::Texture &t, std::ostream &logger=std::cout, const Inventory &inventory=Inventory())

  The constructor.
- bool IsAlive () const

  Checks if the creature is alive.
- const std::string & GetDescription () const
- float TakeHit (float base_damage, const Creature &c2)

  Handles the logic when this creature is attacked.
- float Attack (Creature &c2, const Sword &sword={}) const

  Attacks another creature with a given item.
- virtual void **Update** ()

  Updates state of the creature. Later probably need to pass something cleverer, like game class, tho i am not sure yet.
- void Draw ()

  Draws the creature's sprite in a given window.
- void SetVelocity (const sf::Vector2< float > &newVelocity)

  Setter for velocity.
- void SetVelocityX (float nvx)

  Sets x velocity.
- void SetVelocityY (float nvy)

  Sets y velocity.
- void SetTexture (const sf::Texture &t)

  Sets texture.
- virtual void SetPosition (const sf::Vector2< float > &position)

  Sets position.
- void DrawHealthBar (sf::RenderWindow &window)

  draws the healthbar of the creature
- const sf::Vector2< float > & GetPosition () const
- const Room & GetRoom () const
- Inventory & GetInventory ()

Additional Inherited Members

Protected Member Functions inherited from Creature

- void TakeDamage_ (float damage)

  Reduces health by damage. If damage is greater then health, sets health to 0.
- void **UpdatePosition** ()

  Updates position based on current position, velocity and available rooms. If position + velocity is in one of the available rooms then the position is updated, if not then it stays the same.
- void **UpdateRotation** ()

  Updated rotation so that the creatures faces the direction it is supposed to face, as specified by GetFacingDirection() .
- void TurnToDirection (float dx, float dy)

  Turns the creature to the direction of the vector (dx, dy)

Protected Attributes inherited from Creature

- const float **maxVelocity_**

  max movement speed of the creature\nNote: this is the limit for vx and vy separately.
- const float **maxHealth_**

  max health of the creature
- float **health_**

  current health of the creature.
- int **base_damage_**

  damage that creature deals with no items
- const std::string **type_**

  type of the character (e.g. monster)
- std::string **name_**

  name of the character
- std::string **description_**

  description in the format "Creature of type <type_> named <name>"
- Inventory **inventory_**

  the inventory_ of the creature
- sf::Texture & **texture_**

  texture
- sf::Vector2< float > **position_**

  position of the creature on the screen
- sf::Vector2< float > **velocity_**

  current velocity of the creature
- sf::RenderWindow & **window_**

  window in which the creature is supposed to be drawn
- std::ostream & **logger_**

  stream to log information about the class.
- Room **room_**

  room the creature is in.
- sf::RectangleShape **creatureRect_**

  The rectangle representing the creature.

4.9.1 Detailed Description

Class for the Player.

4.9.2 Member Function Documentation

4.9.2.1 GetItemInUse()

```
int Player::GetItemInUse ( ) const
```

Returns

item in use index

4.9.2.2 GetPosition()

```
sf::Vector2f Player::GetPosition ( )
```

Returns

current position

4.9.2.3 GetRoomIndex()

```
int Player::GetRoomIndex ( ) const
```

Returns

current room index

4.9.2.4 SetItemInUse()

```
void Player::SetItemInUse (
            int index )
```

Sets itemInUse.

Parameters

index	new value for itemInUse

4.9.2.5 SetMonstersCleared()

```
void Player::SetMonstersCleared (
            bool monstersCleared )
```

Set monstersCleared param.

Parameters

monstersCleared	new value for the param

4.9.2.6 SetRoom() [1/2]

```
void Player::SetRoom (
            Room & room )  [override], [virtual]
```

Sets the room.

Parameters

room	new room

Reimplemented from Creature.

4.9.2.7 SetRoom() [2/2]

```
void Player::SetRoom (
            Room & room,
            std::vector< Monster * > & monsters,
            std::vector< sf::Vector2f > & potionPos )
```

Sets the room, if needed spawns the monster and the potions.

Parameters

room	new room
monsters	vector of pointers to monsters
potionPos	vector of positions of health potions

4.9.2.8 SetRooms()

```
void Player::SetRooms (
            const std::vector< Room > & rooms )
```

Sets rooms.

Parameters

rooms	new vector of rooms

4.9.2.9 SpawnMonsters()

```
void Player::SpawnMonsters (
            std::vector< Monster * > & res )
```

Spawns the monsters in the room with the player. If the room is a corridor does nothing.

Parameters

res	vector of pointers to monsters to add the spawned monsters to.

4.9.2.10 SpawnPotion()

```
void Player::SpawnPotion (
```

```
            std::vector< sf::Vector2f > & pos )
```

Spawns potions in the room with the player with predefined probability.

Parameters

pos	vector of positions of health potions to add the spawned potions to.

4.9.2.11 TryAttack()

```
void Player::TryAttack (
            const std::vector< Monster * > & monsters )
```

Tries to attack; if there are no monsters close enough, does nothing; otherwise attacks the closest one.

Parameters

monsters	vector of pointers to monsters

4.9.2.12 TryPickup()

```
void Player::TryPickup (
            std::vector< sf::Vector2f > & potionPositions )
```

Tries to pick up a potion; if there are no potions close enough, does nothing; otherwise picks up the closest one.

Parameters

potionPositions	vector of positions of health potions

4.9.2.13 Update()

```
void Player::Update (
            std::vector< Monster * > & monsters,
            std::vector< sf::Vector2f > potions )
```

Updates the player; if needed spawns the monsters and potions.

Parameters

monsters	vector of pointers to monsters
potions	vector of positions of health potions

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

- Creature/GameCharacter.hpp
- Creature/GameCharacter.cpp

4.10 Room Class Reference

Class representing a Room in the game.

`#include <room.hpp>`

Public Member Functions

- Room (int x, int y, int width, int height, bool isCorridor)

  Constructor for the Room class.
- bool **operator==** (const Room &other) const
- bool **operator!=** (const Room &other) const
- int **GetId** () const
- std::tuple< bool, float, float, sf::Vector2f > IsInside (sf::Vector2f pos, float sz) const

  Checks if the entity is inside the room. Additionally returns multipliers for velocity: if the entity is near the borders of the room, then the needed speed is multiplied by -1.
- bool **IsCorridor** () const
- sf::Vector2f **RandomPos** (float sz) const

Public Attributes

- int x
- int y
- int width
- int height
- std::vector< std::vector< sf::Texture * > > tileTextures

4.10.1 Detailed Description

Class representing a Room in the game.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Room()

```
Room::Room (
            int x,
            int y,
            int width,
            int height,
            bool isCorridor )
```

Constructor for the Room class.

Parameters

x	X-coordinate of the room.
y	Y-coordinate of the room.
width	Width of the room.
height	Height of the room.

4.10.3 Member Function Documentation

4.10.3.1 IsInside()

```
std::tuple< bool, float, float, sf::Vector2f > Room::IsInside (
            sf::Vector2f pos,
            float sz ) const
```

Checks if the entity is inside the room. Additionally returns multipliers for velocity: if the entity is near the borders of the room, then the needed speed is multiplied by -1.

Parameters

pos	position of the entity
radius	size of the entity

Returns

tuple: boolean value (true if the entity is inside) and pair of speed multipliers

4.10.4 Member Data Documentation

4.10.4.1 height

```
int Room::height
```

Height of the room.

4.10.4.2 tileTextures

```
std::vector<std::vector<sf::Texture *> > Room::tileTextures
```

2D vector storing the tile textures of the room.

4.10.4.3 width

```
int Room::width
```

Width of the room.

4.10.4.4 x

```
int Room::x
```

X-coordinate of the room.

4.10.4.5 y

```
int Room::y
```

Y-coordinate of the room.

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

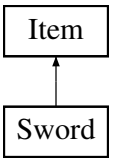
- room.hpp
- room.cpp

4.11 Sword Class Reference

A sword - an item that buffs the damage (multiplies it by some value)

```
#include <Item.hpp>
```

Inheritance diagram for Sword:



Public Member Functions

- **Sword** (const std::string &name, float multiplier)
- float **GetMultiplier** () const

Public Member Functions inherited from Item

- **Item** (const std::string &name, bool isSword)
- const std::string & **GetName** () const
- bool **IsSword** () const

Additional Inherited Members

Static Public Member Functions inherited from Item

- static void Draw (sf::RenderWindow &window, const sf::Vector2f &pos, float maxSize, const sf::Texture &t)

  Draws the item.

Protected Attributes inherited from Item

- std::string **name_**

  name of the item.
- bool **isSword_** {}

  true if the item is a sword, false otherwise.

4.11.1 Detailed Description

A sword - an item that buffs the damage (multiplies it by some value)

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

- Item.hpp

Chapter 5

File Documentation

5.1 Creature.hpp

```
00001 //
00002 // Created by dannypa on 05.11.23.
00003 //
00004
00005 #ifndef DUNGEONCRAWLER_CHARACTER_H
00006 #define DUNGEONCRAWLER_CHARACTER_H
00007
00008 #include <algorithm>
00009 #include <string>
00010 #include <iostream>
00011 #include "../inventory.hpp"
00012 #include "SFML/Graphics.hpp"
00013 #include "../helper.hpp"
00014 #include "../room.hpp"
00015 #include "../textureManager.hpp"
00016
00017 // Constants
00018 const float CREATURE_SIZE = 1.0f;
00019 const float HEALTH_BAR_START_WIDTH = 0.3f;
00020 const float HEALTH_BAR_HEIGHT = 0.2f;
00021
00023 class Creature {
00024  public:
00037   Creature(const std::string &type,
00038            const std::string &name,
00039            int maxHealth,
00040            float maxVelocity,
00041            const sf::Vector2<float> &initialPos,
00042            sf::RenderWindow &window,
00043            const Room &room,
00044            int base_damage,
00045            sf::Texture &t,
00046            std::ostream &logger = std::cout,
00047            const Inventory &inventory = Inventory());
00048
00051   [[nodiscard]] bool IsAlive() const { return health_ > 0; }
00052
00054   [[nodiscard]] const std::string &GetDescription() const;
00055
00060   float TakeHit(float base_damage, const Creature &c2);
00061
00066   float Attack(Creature &c2, const Sword &sword = {}) const;
00067
00070   virtual void Update();
00071
00074   void Draw();
00075
00078   void SetVelocity(const sf::Vector2<float> &newVelocity);
00079
00082   void SetVelocityX(float nvx);
00083
00086   void SetVelocityY(float nvy);
00087
00090   void SetTexture(const sf::Texture &t);
00091
00094   virtual void SetPosition(const sf::Vector2<float> &position);
00095
00098   virtual void SetRoom(Room &room);
```

```
00099
00102   void DrawHealthBar(sf::RenderWindow &window);
00103
00105   const sf::Vector2<float> &GetPosition() const;
00106
00108   const Room &GetRoom() const;
00109
00111   Inventory &GetInventory();
00112
00113  protected:
00115   const float maxVelocity_;
00116
00118   const float maxHealth_;
00119
00121   float health_;
00122
00124   int base_damage_;
00125
00127   const std::string type_;
00128
00130   std::string name_;
00131
00133   std::string description_;
00134
00136   Inventory inventory_;
00137
00139   sf::Texture &texture_;
00140
00142   sf::Vector2<float> position_;
00143
00145   sf::Vector2<float> velocity_;
00146
00148   sf::RenderWindow &window_;
00149
00151   std::ostream &logger_;
00152
00154   Room room_;
00155
00157   sf::RectangleShape creatureRect_;
00158
00161   void TakeDamage_(float damage);
00162
00166   virtual std::vector<Room> GetAvailableRooms();
00167
00170   void UpdatePosition();
00171
00174   void UpdateRotation();
00175
00179   virtual sf::Vector2f GetFacingDirection();
00180
00184   void TurnToDirection(float dx, float dy);
00185 };
00186
00187 #endif //DUNGEONCRAWLER_CHARACTER_H
```

5.2 GameCharacter.hpp

```
00001 //
00002 // Created by dannypa on 13.11.23.
00003 //
00004
00005 #ifndef DUNGEONCRAWLER_SRC_CREATURE_GAMECHARACTERS_HPP_
00006 #define DUNGEONCRAWLER_SRC_CREATURE_GAMECHARACTERS_HPP_
00007 #include <unordered_set>
00008 #include "Creature.hpp"
00009 #include "../room.hpp"
00010 #include "../helper.hpp"
00011
00012 // Constants
00013 // monster behavior
00014 const int TICKS_PER_SECOND = 2;
00015 const long TICK_TIME = CLOCKS_PER_SEC / TICKS_PER_SECOND;
00016 const float MONSTER_ATTACK_RADIUS = 2.f;
00017 const float MONSTER_DIRECTION_CHANGE_PROBABILITY = 0.2f;
00018 // player behavior
00019 const float PLAYER_ATTACK_RADIUS = 2.2f;
00020 // spawn monsters
00021 const int DEFAULT_MONSTER_DAMAGE = 10;
00022 const int DEFAULT_MONSTER_HEALTH = 60;
00023 const float DEFAULT_MONSTER_VELOCITY = 0.085f;
00024
00025 const float BOSS_MONSTER_DAMAGE_MULTIPLIER = 1.5f;
00026 const float BOSS_MONSTER_DAMAGE = DEFAULT_MONSTER_DAMAGE * BOSS_MONSTER_DAMAGE_MULTIPLIER;
```

```
00027 const int BOSS_MONSTER_HEALTH = 250;
00028 const float BOSS_MONSTER_VELOCITY = 0.07f;
00029 // spawn items
00030 const float POTION_SPAWN_PROBABILITY = 0.33f;
00031
00032 class Player;
00033
00035 class Monster : public Creature {
00036  public:
00037   Monster(const std::string &type,
00038           const std::string &name,
00039           int max_health,
00040           float max_velocity,
00041           const sf::Vector2<float> &initial_pos,
00042           sf::RenderWindow &window,
00043           const Room &room,
00044           int base_damage,
00045           sf::Texture &texture,
00046           std::ostream &logger = std::cout,
00047           Inventory inventory = Inventory()) : Creature(type,
00048                                                         name,
00049                                                         max_health,
00050                                                         max_velocity,
00051                                                         initial_pos,
00052                                                         window,
00053                                                         room, base_damage, texture,
00054                                                         logger,
00055                                                         inventory) {}
00056
00060   void tick(Player &p);
00061
00062  private:
00064   long lastTick_ = 0;
00065 };
00066
00068 class Player : public Creature {
00069  public:
00070   Player(const std::string &type, const std::string &name, int maxHealth, float maxVelocity,
00071          const sf::Vector2<float> &initialPos,
00072          sf::RenderWindow &window,
00073          Room room,
00074          sf::Texture &texture,
00075          std::ostream &logger = std::cout,
00076          Inventory inventory = Inventory()) :
00077       Creature(type, name, maxHealth, maxVelocity, initialPos, window, room, 25, texture, logger,
      inventory) {};
00078
00080   int GetRoomIndex() const;
00081
00083   sf::Vector2f GetPosition();
00084
00087   void SetRoom(Room &room) override;;
00088
00093   void SetRoom(Room &room, std::vector<Monster *> &monsters, std::vector<sf::Vector2f> &potionPos);
00094
00098   void Update(std::vector<Monster *> &monsters, std::vector<sf::Vector2f> potions);
00099
00102   void SetRooms(const std::vector<Room> &rooms);
00103
00106   void SetMonstersCleared(bool monstersCleared);
00107
00110   void SetItemInUse(int index);
00111
00113   int GetItemInUse() const;
00114
00117   void TryHealing();
00118
00121   void TryAttack(const std::vector<Monster *> &monsters);
00122
00126   void TryPickup(std::vector<sf::Vector2f> &potionPositions);
00127
00130   void SpawnMonsters(std::vector<Monster *> &res);
00131
00134   void SpawnPotion(std::vector<sf::Vector2f> &pos);
00135
00136  private:
00138   int itemInUse = 0;
00139
00141   std::vector<Room> rooms_;
00142
00144   int roomIndex_;
00145
00147   bool monstersCleared_;
00148
00152   std::vector<Room> GetAvailableRooms() override;
00153
00157   sf::Vector2f GetFacingDirection() override;
```

```
00158
00160   void UpdateRoomIndex();
00161 };
00162
00163 #endif //DUNGEONCRAWLER_SRC_CREATURE_GAMECHARACTERS_HPP_
```

5.3 dungeon.hpp

```
00001 #ifndef DUNGEON_HPP
00002 #define DUNGEON_HPP
00003
00004 #include <vector>
00005 #include "room.hpp"
00006
00007 // Constants
00008 const int GRID_SIZE = 20;
00009 const int ROOM_MIN_SIZE = 10;
00010 const int ROOM_MAX_SIZE = 20;
00011
00015 enum Direction {
00016   UP,
00017   DOWN,
00018   LEFT,
00019   RIGHT
00020 };
00021
00026 class Dungeon {
00027  public:
00031   Dungeon();
00032   ~Dungeon();
00033
00045   void GenerateDungeon(std::vector<Room> &rooms,
00046                        std::vector<Room> &corridors,
00047                        int numRooms,
00048                        int TILE_SIZE,
00049                        int WINDOW_WIDTH,
00050                        int WINDOW_HEIGHT);
00051
00052  private:
00053   std::vector<std::vector<bool>> roomGrid_;
00054
00069   void GenerateRooms(std::vector<Room> &rooms,
00070                      std::vector<Room> &corridors,
00071                      int numRooms,
00072                      int x,
00073                      int y,
00074                      float GRID_WIDTH,
00075                      float GRID_HEIGHT,
00076                      int WINDOW_WIDTH,
00077                      int WINDOW_HEIGHT,
00078                      int GRID_SIZE);
00079 };
00080
00081 #endif
```

5.4 game.hpp

```
00001 #ifndef GAME_HPP
00002 #define GAME_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <SFML/Audio.hpp>
00006 #include <vector>
00007 #include "dungeon.hpp"
00008 #include "inventory.hpp"
00009 #include "textureManager.hpp"
00010 #include "Creature/GameCharacter.hpp"
00011 #include "gameOverScreen.hpp"
00012
00013 // Constants
00014 // game
00015 const std::string WINDOW_TITLE = "Dungeon Crawler";
00016 const unsigned int WINDOW_WIDTH = 1080u;
00017 const unsigned int WINDOW_HEIGHT = 720u;
00018 const int FRAMERATE_LIMIT = 60;
00019 const float ZOOM_LEVEL = 50.0f;
00020 // dungeon
00021 const int ROOM_AMOUNT = 5;
00022 const int TILE_SIZE = 1;
```

```
00023 // player
00024 const float PLAYER_WALKING_SPEED = 0.12f;
00025 const float PLAYER_RUNNING_SPEED = 0.16f;
00026 // items
00027 const float START_SWORD_MULTIPLIER = 1.2f;
00028 const int INITIAL_POTION_NUMBER = 3;
00029
00030 // Enum for room types
00031 enum RoomType {
00032   ROOM,
00033   CORRIDOR,
00034 };
00035
00039 class Game {
00040  public:
00041   Game();
00042   ~Game();
00043   void Run();
00044
00045  private:
00047   Player player_;
00048   std::vector<Monster *> monsters_;
00049   std::vector<sf::Vector2f> potions_;
00050
00051   sf::RenderWindow window_;
00052
00053   Dungeon dungeon_;
00054   Inventory inventory_;
00055   GameOverScreen gameOverScreen_;
00056   std::vector<Room> rooms_;
00057   std::vector<Room> corridors_;
00058
00059   // Movement flags
00060   bool moveUp_ = false;
00061   bool moveDown_ = false;
00062   bool moveLeft_ = false;
00063   bool moveRight_ = false;
00064   bool isRunning_ = false;
00065   bool gameWon_ = false;
00066   bool gameLost_ = false;
00067
00068   // Methods
00069   static void InitializeTextures();
00070   bool CheckWinning(bool monstersKilled);
00071   bool CheckLosing();
00072   void InitializeWindow();
00073   void ProcessEvents();
00074   void Update();
00075   void Render();
00076   void InitiateDungeon();
00077   void InitiateInventory();
00078   void DrawDungeon();
00079   void DrawRoom(const Room &room, RoomType type);
00080 };
00081
00082 #endif /* GAME_HPP */
```

5.5 gameOverScreen.hpp

```
00001 #include <SFML/Graphics.hpp>
00002
00003 // Constants
00004 const int VERDICT_CHARACTER_SIZE = 100;
00005 const int PLAY_AGAIN_CHARACTER_SIZE = 24;
00006 const float BUTTON_SCALE = 0.02f;
00007 const float NEW_TEXT_SCALE = 0.01f;
00008 const int BUTTON_WIDTH = 200;
00009 const int BUTTON_HEIGHT = 50;
00010
00014 class GameOverScreen {
00015  public:
00024   void Render(sf::RenderWindow &window, sf::Font &font, std::string text, sf::Color textColor,
      sf::Color buttonColor);
00025
00030   sf::RectangleShape GetPlayAgainButton();
00031
00032  private:
00033   sf::RectangleShape playAgainButton_;
00044   static sf::Text CreateText(sf::Font &font,
00045                              const std::string &text,
00046                              sf::Color textColor,
00047                              sf::RenderWindow &window,
00048                              unsigned int characterSize);
```

```
00049
00057   static sf::RectangleShape CreatePlayAgainButton(sf::RenderWindow &window,
00058                                                    sf::Text &buttonText,
00059                                                    sf::Color buttonColor);
00060
00068   static void DrawElements(sf::RenderWindow &window,
00069                            sf::Text &winText,
00070                            sf::RectangleShape &playAgainButton,
00071                            sf::Text &buttonText);
00072 };
```

5.6 helper.hpp

```
00001 //
00002 // Created by dannypa on 20.11.23.
00003 //
00004
00005 #ifndef DUNGEONCRAWLER_SRC_HELPER_HPP_
00006 #define DUNGEONCRAWLER_SRC_HELPER_HPP_
00007 #include "iostream"
00008 #include "SFML/Graphics.hpp"
00009 #include "random"
00010
00018 template<typename T>
00019 T bound(T x, T lower, T upper, T EPS) {
00020   x = std::max(lower + EPS, x);
00021   x = std::min(x, upper - EPS);
00022   return x;
00023 }
00024
00025 namespace help {
00030 template<typename T>
00031 T square(const sf::Vector2<T> v) {
00032   return v.x * v.x + v.y * v.y;
00033 }
00034
00039 template<typename T>
00040 double len(const sf::Vector2<T> v) {
00041   return sqrt(square(v));
00042 }
00043
00048 template<typename T>
00049 bool close(T a, T b, T EPS) {
00050   return std::abs(a - b) < EPS;
00051 }
00052
00058 template<typename T>
00059 bool close(const sf::Vector2<T> &v1, const sf::Vector2<T> &v2, T maxDist) {
00060   return square(v1 - v2) <= maxDist * maxDist;
00061 }
00062 }
00063
00064 #endif //DUNGEONCRAWLER_SRC_HELPER_HPP_
00065
```

5.7 inventory.hpp

```
00001 #ifndef INVENTORY_HPP
00002 #define INVENTORY_HPP
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <unordered_map>
00007 #include <string>
00008 #include <algorithm>
00009 #include <SFML/Graphics.hpp>
00010 #include "Item.hpp"
00011 #include "textureManager.hpp"
00012
00013 // Constants
00014 const float ITEM_CIRCLE_RADIUS = 0.5f;
00015 const float ITEM_SIZE = 2 * ITEM_CIRCLE_RADIUS;
00016 const float ITEM_CIRCLE_OUTLINE_THICKNESS = 0.1f;
00017 const sf::Color ITEM_IN_USE_COLOR = sf::Color::Red;
00018 const sf::Color ITEM_NOT_IN_USE_COLOR = sf::Color::Blue;
00019 const int QUANTITY_TEXT_SCALE_X = 10;
00020 const int QUANTITY_TEXT_SCALE_Y = 20;
00021
00028 class Inventory {
```

```
00029  private:
00031   std::unordered_map<std::string, int> counter_;
00032
00035   std::vector<std::pair<bool, int» isSword_;
00036
00038   std::vector<Sword> swords_;
00039
00041   std::vector<HealthPotion> potions_;
00042
00048   template<typename T>
00049   void AddItem(const T &item, std::vector<T> &itemsVector, int quantity);
00050
00060   template<typename T>
00061   float DrawItems(std::vector<T> items,
00062                   const sf::Texture &t,
00063                   float x,
00064                   float y,
00065                   sf::RenderWindow &window,
00066                   bool itemInUse);
00067
00068  public:
00069
00073   void AddSword(const Sword &item, int quantity) {
00074     AddItem(item, swords_, quantity);
00075   }
00076
00080   void AddPotion(const HealthPotion &item, int quantity) {
00081     AddItem(item, potions_, quantity);
00082   }
00083
00088   int GetSize();
00089
00093   void Draw(sf::RenderWindow &window, int itemInUse);
00094
00097   Sword GetSword(int index);
00098
00101   HealthPotion GetPotion(int index);
00102
00105   int GetHealingAmount(int index);
00106
00109   bool IsSword(int i) { return isSword_[i].first; }
00110 };
00111
00112 template<typename T>
00113 void Inventory::AddItem(const T &item, std::vector<T> &itemsVector, int quantity) {
00114   if (counter_.find(item.GetName()) == counter_.end()) {
00115     counter_[item.GetName()] = 0;
00116     isSword_.emplace_back(item.IsSword(), itemsVector.size());
00117     itemsVector.push_back(item);
00118   }
00119   counter_[item.GetName()] += quantity;
00120 }
00121 #endif // INVENTORY_HPP
```

5.8 Item.hpp

```
00001 #ifndef DUNGEONCRAWLER_SRC_ITEM_HPP_
00002 #define DUNGEONCRAWLER_SRC_ITEM_HPP_
00003
00004 #include <iostream>
00005 #include "string"
00006 #include <SFML/Graphics.hpp>
00007
00008 // Constants
00009 const int DEFAULT_HEAL_AMOUNT = 10;
00010
00012 class Item {
00013  public:
00014   Item() = default;
00015   Item(const std::string &name, bool isSword) : name_(name), isSword_(isSword) {};
00016
00017   [[nodiscard]] const std::string &GetName() const { return name_; }
00018
00019   [[nodiscard]] bool IsSword() const { return isSword_; }
00020
00026   static void Draw(sf::RenderWindow &window,
00027                    const sf::Vector2f &pos,
00028                    float maxSize,
00029                    const sf::Texture &t);
00030  protected:
00032   std::string name_;
00033
00035   bool isSword_{};
```

```
00036 };
00037
00039 class Sword : public Item {
00040  public:
00041   Sword() = default;
00042   Sword(const std::string &name, float multiplier) : Item(name, true), multiplier_(multiplier) {};
00043
00044   [[nodiscard]] float GetMultiplier() const { return multiplier_; }
00045  private:
00047   float multiplier_ = 1;
00048 };
00049
00051 class HealthPotion : public Item {
00052  public:
00053   HealthPotion() = default;
00054   explicit HealthPotion(float hpRestored) : Item("Health potion " + std::to_string(hpRestored),
      false),
00055                                             hpRestored_(hpRestored) {};
00056
00057   [[nodiscard]] float GetHpRestored() const { return hpRestored_; };
00058  private:
00060   float hpRestored_ = DEFAULT_HEAL_AMOUNT;
00061 };
00062 #endif //DUNGEONCRAWLER_SRC_ITEM_HPP_
```

5.9 room.hpp

```
00001 #ifndef ROOM_HPP
00002 #define ROOM_HPP
00003
00004 #include <SFML/Graphics.hpp>
00005 #include <vector>
00006 #include <cstdlib>
00007 #include <ctime>
00008 #include <iostream>
00009 #include "textureManager.hpp"
00010 #include "helper.hpp"
00011
00012 // Constants
00013 const float ROOM_INSIDE_EPS = 0.1;
00014 const float ROOM_BOUND_EPS = 0.9f * ROOM_INSIDE_EPS;
00015
00019 class Room {
00020  public:
00029   Room(int x, int y, int width, int height, bool isCorridor);
00030
00031   ~Room();
00032
00033   int x;
00034   int y;
00035   int width;
00036   int height;
00037   bool operator==(const Room &other) const { return id_ == other.id_; }
00038   bool operator!=(const Room &other) const { return id_ != other.id_; }
00039
00040   [[nodiscard]] int GetId() const { return id_; };
00041
00047   [[nodiscard]] std::tuple<bool, float, float, sf::Vector2f> IsInside(sf::Vector2f pos, float sz)
      const;
00048
00049   std::vector<std::vector<sf::Texture *» tileTextures;
00051   [[nodiscard]] bool IsCorridor() const { return isCorridor_; }
00052
00053   [[nodiscard]] sf::Vector2f RandomPos(float sz) const;
00054
00055  private:
00057   bool isCorridor_;
00059   int id_;
00060
00061   static int RandInt(int a, int b);
00062 };
00063
00064 #endif
```

5.10 test.hpp

```
00001 #include <iostream>
00002 #include <SFML/Graphics.hpp>
00003 #include <SFML/Audio.hpp>
```

```
00004 #include <vector>
00005 #include "../dungeon.hpp"
00006 #include "../textureManager.hpp"
00007 #include "../Creature/Creature.hpp"
00008 #include "../Creature/GameCharacter.hpp"
00009
00010 bool testDungeonGeneration() {
00011   // Test dungeon_ generation with 5 rooms_
00012   int numRooms = 5;
00013   int TILE_SIZE = 32; // Example tile size
00014   int WINDOW_WIDTH = 800; // Example window width
00015   int WINDOW_HEIGHT = 600; // Example window height
00016
00017   std::vector<Room> rooms;
00018   std::vector<Room> corridors;
00019   sf::Texture player_t;
00020
00021   Dungeon dungeon;
00022   dungeon.GenerateDungeon(rooms, corridors, numRooms, TILE_SIZE, WINDOW_WIDTH, WINDOW_HEIGHT);
00023
00024   // Check if the number of generated rooms_ matches the expected number
00025   if (rooms.size() != numRooms) {
00026     std::cout « "Test failed: Incorrect number of rooms_ generated." « std::endl;
00027     return false;
00028   }
00029
00030   // Check if the number of generated corridors_ matches the expected number
00031   if (corridors.size() != --numRooms) {
00032     std::cout « "Test failed: Incorrect number of corridors_ generated." « std::endl;
00033     return false;
00034   }
00035
00036   sf::RenderWindow test_window;
00037   // Create a test creature
00038   Monster test_creature("test", "creature", 50, 10.0f,
00039                         sf::Vector2f(0, 0),
00040                         test_window, Room(0, 0, 0, 0, false), 10, player_t);
00041
00042   // Check if the position matches the expected postition
00043   if (test_creature.GetPosition() != sf::Vector2f(0, 0)) {
00044     std::cout « "Test failed: Incorrect initial postition of test creature." « std::endl;
00045     return false;
00046   }
00047
00048   // Add health postions to the test creatures inventory_
00049   test_creature.GetInventory().AddPotion(HealthPotion(), 3);
00050
00051   // Check if the size of the inventory_ matches the expected size
00052
00053   if (test_creature.GetInventory().GetSize() != 1) {
00054     std::cout « "Test failed: Incorrect inventory_ size." « std::endl;
00055     return false;
00056   }
00057
00058   // All tests passed
00059   std::cout « "All tests passed\n";
00060   return true;
00061 }
00062
00063 int runTests() {
00064   // Run tests
00065   if (!testDungeonGeneration()) {
00066     std::cout « "Tests failed!\n";
00067     return 1;
00068   }
00069
00070   return 0;
00071 }
```

5.11 textureManager.hpp

```
00001 #include <SFML/Graphics.hpp>
00002
00003 extern sf::Texture player_t;
00004 extern sf::Texture assassin_t;
00005 extern sf::Texture room_t1;
00006 extern sf::Texture room_t2;
00007 extern sf::Texture corridor_t1;
00008 extern sf::Texture sword_inv_t;
00009 extern sf::Texture potion_inv_t;
00010 extern sf::Texture boss_t;
00011 extern sf::Font font;
```

Index