

Maze Game

1.0

Generated by Doxygen 1.9.8

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Game Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 Game()	6
3.1.3 Member Function Documentation	7
3.1.3.1 handleInput()	7
3.1.3.2 initialize()	7
3.1.3.3 render()	8
3.1.3.4 run()	9
3.1.3.5 update()	10
3.1.4 Member Data Documentation	10
3.1.4.1 generator	10
3.1.4.2 isRunning	10
3.1.4.3 maze	11
3.1.4.4 player	11
3.1.4.5 renderer	11
3.2 Generator Class Reference	11
3.2.1 Detailed Description	11
3.2.2 Constructor & Destructor Documentation	12
3.2.2.1 Generator()	12
3.2.3 Member Function Documentation	12
3.2.3.1 carvePassages()	12
3.2.3.2 generate()	13
3.3 Maze Class Reference	14
3.3.1 Detailed Description	15
3.3.2 Constructor & Destructor Documentation	15
3.3.2.1 Maze() [1/2]	15
3.3.2.2 Maze() [2/2]	15
3.3.3 Member Function Documentation	15
3.3.3.1 getCell()	15
3.3.3.2 getExitCoordinates()	16
3.3.3.3 getSize()	17
3.3.3.4 isInMazeBounds()	17
3.3.3.5 isWall()	18
3.3.3.6 setCell()	18
3.3.3.7 setExitCoordinates()	19

3.3.3.8 setSize()	19
3.3.4 Member Data Documentation	19
3.3.4.1 exitCoordinates	19
3.3.4.2 maze	20
3.3.4.3 size	20
3.4 Player Class Reference	20
3.4.1 Detailed Description	21
3.4.2 Constructor & Destructor Documentation	21
3.4.2.1 Player() [1/2]	21
3.4.2.2 Player() [2/2]	21
3.4.3 Member Function Documentation	21
3.4.3.1 getCurrentPosition()	21
3.4.3.2 getX()	21
3.4.3.3 getY()	22
3.4.3.4 go()	22
3.4.3.5 setCurrentPosition()	23
3.4.4 Member Data Documentation	24
3.4.4.1 currentPosition	24
3.5 Renderer Class Reference	24
3.5.1 Detailed Description	24
3.5.2 Constructor & Destructor Documentation	25
3.5.2.1 Renderer()	25
3.5.3 Member Function Documentation	25
3.5.3.1 clearScreen()	25
3.5.3.2 draw()	25
3.5.3.3 showTutorial()	26
4 File Documentation	29
4.1 game.cpp File Reference	29
4.1.1 Detailed Description	30
4.1.2 Function Documentation	30
4.1.2.1 customChoice()	30
4.1.2.2 helloFunction()	31
4.2 game.hpp File Reference	31
4.3 game.hpp	32
4.4 generator.cpp File Reference	33
4.4.1 Detailed Description	34
4.5 generator.hpp File Reference	34
4.6 generator.hpp	35
4.7 helper.cpp File Reference	35
4.7.1 Detailed Description	36
4.7.2 Function Documentation	36

4.7.2.1 printHelp()	36
4.8 helper.hpp File Reference	37
4.8.1 Function Documentation	37
4.8.1.1 printHelp()	37
4.9 helper.hpp	37
4.10 main.cpp File Reference	38
4.10.1 Function Documentation	38
4.10.1.1 main()	38
4.11 maze.cpp File Reference	39
4.11.1 Detailed Description	39
4.12 maze.hpp File Reference	40
4.13 maze.hpp	40
4.14 player.cpp File Reference	41
4.14.1 Detailed Description	42
4.15 player.hpp File Reference	42
4.16 player.hpp	43
4.17 renderer.cpp File Reference	43
4.17.1 Detailed Description	44
4.18 renderer.hpp File Reference	44
4.18.1 Macro Definition Documentation	46
4.18.1.1 ANSI_CLEAR	46
4.19 renderer.hpp	46
Index	47

Chapter 1

Class Index

1.1 Class List

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

Game	High-level controller of the maze game	5
Generator	Generates random mazes using the recursive backtracking algorithm	11
Maze	Represents a maze structure	14
Player	Represents the player in the maze	20
Renderer	Handles console output for the maze game	24

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

game.cpp	Implementation of the Game class and helper menu functions	29
game.hpp	31
generator.cpp	Implementation of the maze generator based on recursive backtracking	33
generator.hpp	34
helper.cpp	Implementation of small helper utilities for the game	35
helper.hpp	37
main.cpp	38
maze.cpp	Implementation of the Maze class	39
maze.hpp	40
player.cpp	Implementation of the Player class	41
player.hpp	42
renderer.cpp	Implementation of the Renderer class	43
renderer.hpp	44

Chapter 3

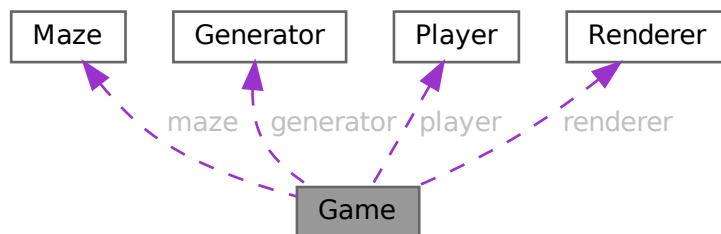
Class Documentation

3.1 Game Class Reference

High-level controller of the maze game.

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

Collaboration diagram for Game:



Public Member Functions

- `Game ()`
Constructs a `Game` object with an initial stopped state.
- `void initialize ()`
Initializes the game before the main loop starts.
- `void run ()`
Runs the main game loop.

Private Member Functions

- void [handleInput](#) ()
Reads and processes a single user input command.
- void [update](#) ()
Updates the game state after a user action.
- void [render](#) () const
Renders the current game state to the terminal.

Private Attributes

- [Maze](#) [maze](#)
Maze currently being played.
- [Generator](#) [generator](#)
Maze generator implementing recursive backtracking.
- [Player](#) [player](#)
Player controlled by the user.
- [Renderer](#) [renderer](#)
Renderer responsible for drawing the maze and player.
- bool [isRunning](#)
Flag indicating whether the main game loop should continue running.

3.1.1 Detailed Description

High-level controller of the maze game.

The [Game](#) class owns all main components ([Maze](#), [Generator](#), [Player](#), [Renderer](#)) and manages the game loop. It is responsible for initializing the game state, handling user input, updating the game logic and rendering the maze.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Game()

```
Game::Game ( )
```

Constructs a [Game](#) object with an initial stopped state.

The actual maze, player and generator are set up in [initialize\(\)](#).

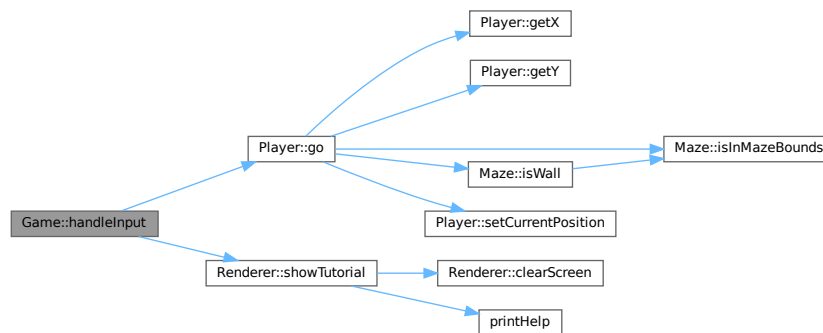
3.1.3 Member Function Documentation

3.1.3.1 handleInput()

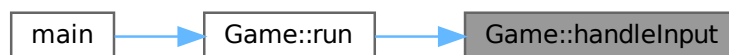
```
void Game::handleInput ( ) [private]
```

Reads and processes a single user input command.

Handles quitting the game, showing the tutorial, or moving the player in the maze according to the input character. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 initialize()

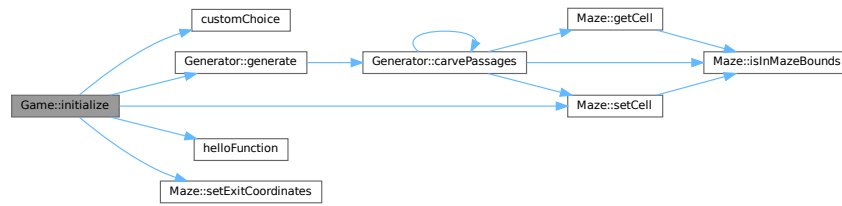
```
void Game::initialize ( )
```

Initializes the game before the main loop starts.

Shows a menu that lets the user choose a predefined maze size or enter a custom size, then:

- creates a [Maze](#) of the chosen dimensions,
- generates a random maze using the [Generator](#),
- places the exit and the player,
- sets `isRunning` to true.

Here is the call graph for this function:



Here is the caller graph for this function:

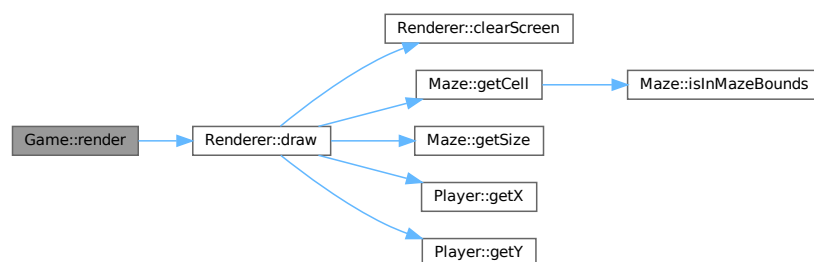


3.1.3.3 render()

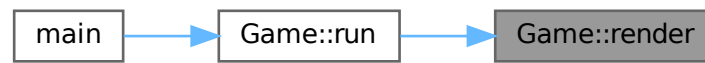
```
void Game::render ( ) const [private]
```

Renders the current game state to the terminal.

Delegates drawing to the [Renderer](#) instance, which prints the maze and the player's position. Here is the call graph for this function:



Here is the caller graph for this function:

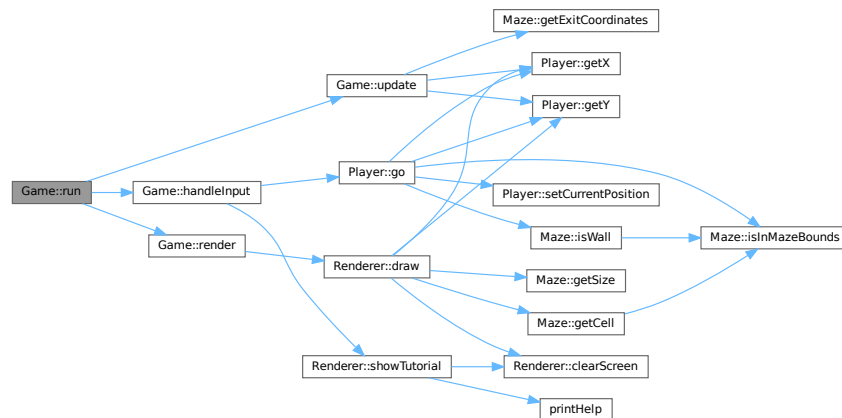


3.1.3.4 run()

```
void Game::run ( )
```

Runs the main game loop.

As long as `isRunning` is true, the loop repeatedly renders the maze, handles user input and updates the game state. Here is the call graph for this function:



Here is the caller graph for this function:

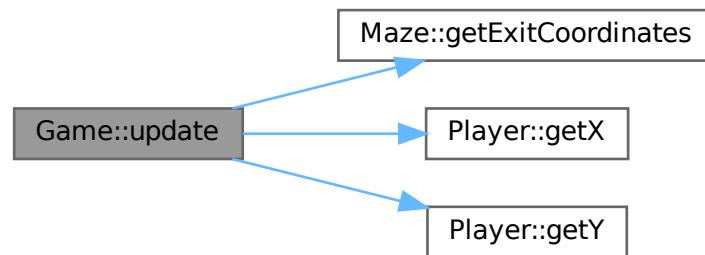


3.1.3.5 update()

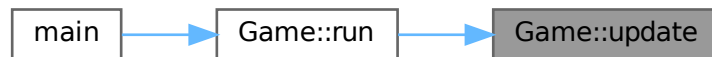
```
void Game::update ( ) [private]
```

Updates the game state after a user action.

Currently, checks whether the player has reached the exit cell and stops the game if the maze is solved. Here is the call graph for this function:



Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 generator

```
Generator Game::generator [private]
```

[Maze](#) generator implementing recursive backtracking.

3.1.4.2 isRunning

```
bool Game::isRunning [private]
```

Flag indicating whether the main game loop should continue running.

3.1.4.3 maze

`Maze` `Game::maze` [private]

`Maze` currently being played.

3.1.4.4 player

`Player` `Game::player` [private]

`Player` controlled by the user.

3.1.4.5 renderer

`Renderer` `Game::renderer` [private]

`Renderer` responsible for drawing the maze and player.

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

- [game.hpp](#)
- [game.cpp](#)

3.2 Generator Class Reference

Generates random mazes using the recursive backtracking algorithm.

```
#include <generator.hpp>
```

Public Member Functions

- [Generator](#) ()
Default constructor.
- void [generate](#) ([Maze](#) &maze, int startX, int startY)
Generates a maze starting from the given cell.

Private Member Functions

- void [carvePassages](#) ([Maze](#) &maze, int x, int y)
Recursively carves passages in the maze.

3.2.1 Detailed Description

Generates random mazes using the recursive backtracking algorithm.

The [Generator](#) operates directly on a [Maze](#) instance and carves passages starting from a given starting cell. The resulting maze is a so-called "perfect maze" (there is exactly one path between any two cells).

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Generator()

```
Generator::Generator ( )
```

Default constructor.

Creates an empty [Generator](#) instance ready to generate mazes.

Initializes an empty [Generator](#) object ready for maze generation.

3.2.3 Member Function Documentation

3.2.3.1 carvePassages()

```
void Generator::carvePassages (
    Maze & maze,
    int x,
    int y ) [private]
```

Recursively carves passages in the maze.

Recursively carves passages from a given cell.

Marks the current cell as a passage and then visits neighbouring cells in random order. For each unvisited neighbour that is within bounds, it removes the wall between the current cell and the neighbour and continues recursively from that neighbour.

Parameters

<i>maze</i>	Reference to the maze being generated.
<i>x</i>	Current cell x-coordinate (column index).
<i>y</i>	Current cell y-coordinate (row index).

The algorithm:

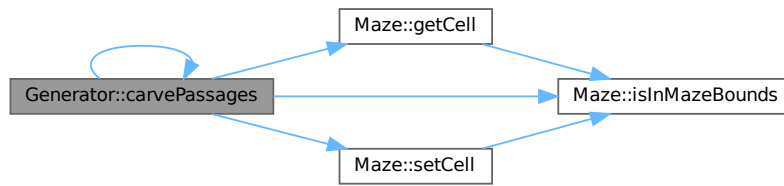
- marks the current cell as a passage ('.'),
- creates a list of four possible directions (up, down, left, right),
- shuffles the directions randomly,
- for each direction computes the neighbour two cells away,
- if the neighbour is inside the maze and still a wall ('#'), removes the wall between current cell and the neighbour and continues recursively from the neighbour.

This produces a connected maze with no cycles (perfect maze).

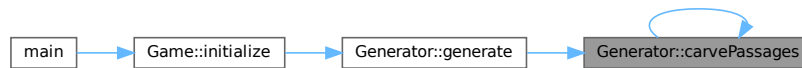
Parameters

<i>maze</i>	Reference to the maze being modified.
<i>x</i>	Current cell x-coordinate.
<i>y</i>	Current cell y-coordinate.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3.2 generate()

```

void Generator::generate (
    Maze & maze,
    int startX,
    int startY )
  
```

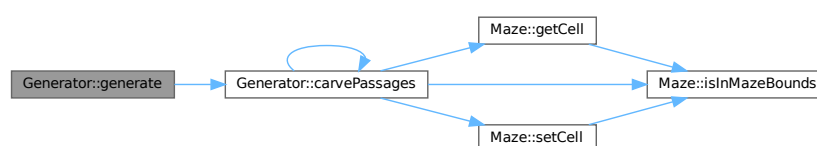
Generates a maze starting from the given cell.

Initializes the recursive backtracking algorithm from the starting position. For correct behaviour of the algorithm, both `startX` and `startY` should be odd coordinates that lie inside the maze.

Parameters

<i>maze</i>	Reference to the maze that will be filled with passages.
<i>startX</i>	Starting cell x-coordinate (should be odd).
<i>startY</i>	Starting cell y-coordinate (should be odd).

Here is the call graph for this function:



Here is the caller graph for this function:



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

- [generator.hpp](#)
- [generator.cpp](#)

3.3 Maze Class Reference

Represents a maze structure.

```
#include <maze.hpp>
```

Public Member Functions

- [Maze](#) ()
- [Maze](#) (int width, int height)
Constructs a maze with specified dimesnions.
- std::pair< int, int > [getSize](#) () const
Gets the size of the maze.
- std::pair< int, int > [getExitCoordinates](#) () const
Gets the exit coordinates.
- char [getCell](#) (int x, int y) const
Gets a cell value at the specified position.
- void [setSize](#) (int width, int height)
- void [setExitCoordinates](#) (int x, int y)
- void [setCell](#) (int x, int y, char value)
Sets a cell value at the specified position.
- bool [isInMazeBounds](#) (int x, int y) const
Checks if coordinates are within maze bounds.
- bool [isWall](#) (int x, int y) const
Checks if a cell is a wall.

Private Attributes

- std::pair< int, int > [size](#)
Width and height of the maze.
- std::vector< std::vector< char > > [maze](#)
2D grid representing the maze structure
- std::pair< int, int > [exitCoordinates](#)
Coordinates of the maze exit.

3.3.1 Detailed Description

Represents a maze structure.

This class manages the maze structure, including its size, walls, passages, and exit coordinates. It provides methods to check boundaries and wall positions

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Maze() [1/2]

```
Maze::Maze ( )
```

3.3.2.2 Maze() [2/2]

```
Maze::Maze (
    int width,
    int height )
```

Constructs a maze with specified dimensions.

Parameters

<i>width</i>	The width of the maze
<i>height</i>	The height of the maze

3.3.3 Member Function Documentation

3.3.3.1 getCell()

```
char Maze::getCell (
    int x,
    int y ) const
```

Gets a cell value at the specified position.

Parameters

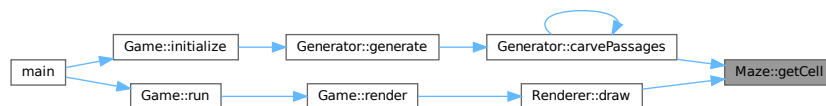
<i>x</i>	X-coordinate
<i>y</i>	Y-coordinate

Returns

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.2 getExitCoordinates()

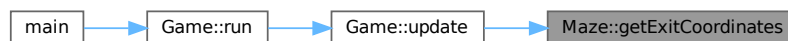
```
std::pair< int, int > Maze::getExitCoordinates ( ) const
```

Gets the exit coordinates.

Returns

A pair containing x and y coordinates of the exit

Here is the caller graph for this function:



3.3.3.3 getSize()

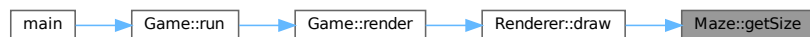
```
std::pair< int, int > Maze::getSize ( ) const
```

Gets the size of the maze.

Returns

A pair containing width and height

Here is the caller graph for this function:



3.3.3.4 isInMazeBounds()

```
bool Maze::isInMazeBounds (
    int x,
    int y ) const
```

Checks if coordinates are within maze bounds.

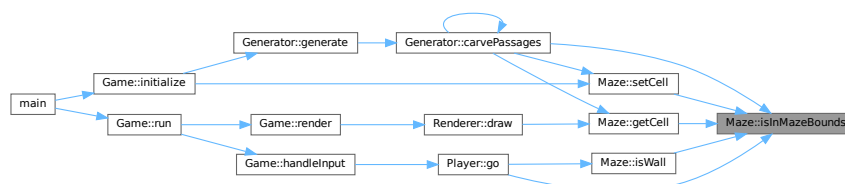
Parameters

<i>x</i>	X-coordinate to check
<i>y</i>	Y-coordinate to check

Returns

true if coordinates are inside the maze, false otherwise

Here is the caller graph for this function:



3.3.3.5 isWall()

```
bool Maze::isWall (
    int x,
    int y ) const
```

Checks if a cell is a wall.

Parameters

<i>x</i>	X-coordinate
<i>y</i>	Y-coordinate

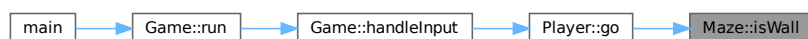
Returns

true if the cell is a wall, false otherwise

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.6 setCell()

```
void Maze::setCell (
    int x,
    int y,
    char value )
```

Sets a cell value at the specified position.

Parameters

<i>x</i>	X-coordinate
<i>y</i>	Y-coordinate
<i>value</i>	Character representing the cell (e.g., "#" for wall, "." for passage)

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.7 setExitCoordinates()

```

void Maze::setExitCoordinates (
    int x,
    int y )
  
```

Here is the caller graph for this function:



3.3.3.8 setSize()

```

void Maze::setSize (
    int width,
    int height )
  
```

3.3.4 Member Data Documentation

3.3.4.1 exitCoordinates

```
std::pair<int, int> Maze::exitCoordinates [private]
```

Coordinates of the maze exit.

3.3.4.2 maze

```
std::vector<std::vector<char> > Maze::maze [private]
```

2D grid representing the maze structure

3.3.4.3 size

```
std::pair<int, int> Maze::size [private]
```

Width and height of the maze.

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

- [maze.hpp](#)
- [maze.cpp](#)

3.4 Player Class Reference

Represents the player in the maze.

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

Public Member Functions

- [Player](#) ()
Default constructor.
- [Player](#) (int x, int y)
Constructs a player at a given position.
- int [getX](#) () const
Gets the current x-coordinate of the player.
- int [getY](#) () const
Gets the current y-coordinate of the player.
- std::pair< int, int > [getCurrentPosition](#) () const
Gets the current player position as a pair.
- void [setCurrentPosition](#) (int x, int y)
Sets the current position of the player.
- void [go](#) (char direction, const [Maze](#) &maze)
Moves the player in the maze according to a direction key.

Private Attributes

- std::pair< int, int > [currentPosition](#)
Current (x, y) coordinates of the player.

3.4.1 Detailed Description

Represents the player in the maze.

The [Player](#) stores its current position as grid coordinates and provides methods to query and update this position. Movement is constrained by the [Maze](#): the player cannot leave the bounds or walk through walls.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 `Player()` [1/2]

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

Default constructor.

Initializes the player at position (0, 0).

3.4.2.2 `Player()` [2/2]

```
Player::Player (
    int x,
    int y )
```

Constructs a player at a given position.

Parameters

<i>x</i>	Initial x-coordinate.
<i>y</i>	Initial y-coordinate.

3.4.3 Member Function Documentation

3.4.3.1 `getCurrentPosition()`

```
std::pair< int, int > Player::getCurrentPosition ( ) const
```

Gets the current player position as a pair.

Returns

Pair (x, y) with current coordinates.

3.4.3.2 `getX()`

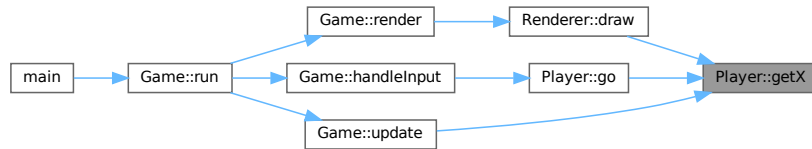
```
int Player::getX ( ) const
```

Gets the current x-coordinate of the player.

Returns

Current x-coordinate.

Here is the caller graph for this function:

**3.4.3.3 getY()**

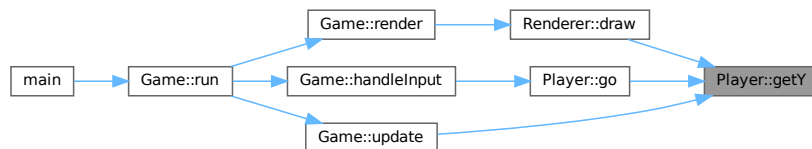
```
int Player::getY ( ) const
```

Gets the current y-coordinate of the player.

Returns

Current y-coordinate.

Here is the caller graph for this function:

**3.4.3.4 go()**

```
void Player::go (
    char direction,
    const Maze & maze )
```

Moves the player in the maze according to a direction key.

Interprets the given character as one of the movement commands:

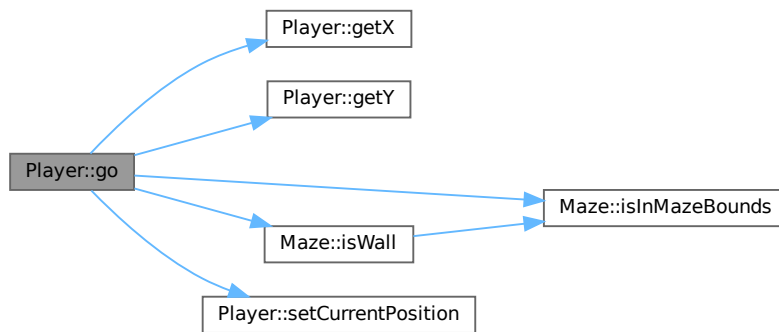
- 'W'/'w' – up,
- 'S'/'s' – down,
- 'A'/'a' – left,
- 'D'/'d' – right.

The player is only moved if the target cell is inside the maze and is not a wall. Any other character is ignored.

Parameters

<i>direction</i>	Character representing the move command.
<i>maze</i>	Constant reference to the Maze used for bounds and wall checks.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.5 setCurrentPosition()

```
void Player::setCurrentPosition (
    int x,
    int y )
```

Sets the current position of the player.

Parameters

<i>x</i>	New x-coordinate.
<i>y</i>	New y-coordinate.

Here is the caller graph for this function:



3.4.4 Member Data Documentation

3.4.4.1 currentPosition

```
std::pair<int,int> Player::currentPosition [private]
```

Current (x, y) coordinates of the player.

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

- [player.hpp](#)
- [player.cpp](#)

3.5 Renderer Class Reference

Handles console output for the maze game.

```
#include <renderer.hpp>
```

Public Member Functions

- [Renderer](#) ()
Default constructor. No initialization required.
- void [draw](#) (const [Maze](#) &maze, const [Player](#) &player) const
Draws the complete maze with player position.
- void [showTutorial](#) () const
Shows the game tutorial/help screen.

Private Member Functions

- void [clearScreen](#) () const
Clears the terminal screen using ANSI escape sequence.

3.5.1 Detailed Description

Handles console output for the maze game.

Uses ANSI escape sequences to clear screen and render the maze grid with the player position overlaid. Also displays the help/tutorial.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Renderer()

```
Renderer::Renderer ( )
```

Default constructor. No initialization required.

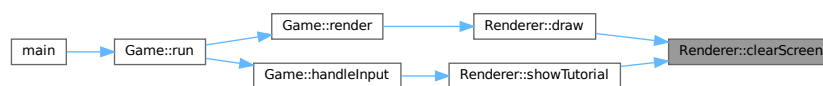
3.5.3 Member Function Documentation

3.5.3.1 clearScreen()

```
void Renderer::clearScreen ( ) const [inline], [private]
```

Clears the terminal screen using ANSI escape sequence.

Sends the standard ANSI clear screen and move cursor to top-left sequence: ESC[2J ESC[H. Here is the caller graph for this function:



3.5.3.2 draw()

```
void Renderer::draw (
    const Maze & maze,
    const Player & player ) const
```

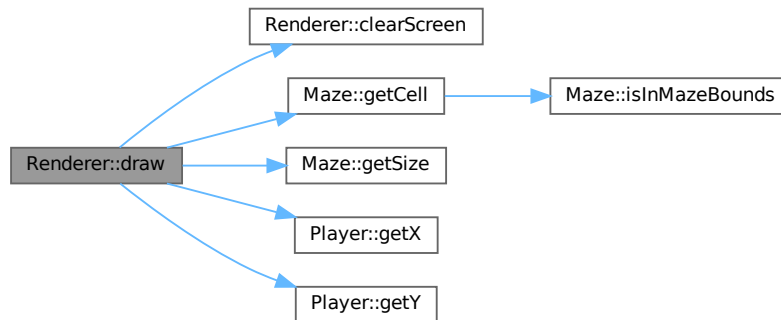
Draws the complete maze with player position.

Clears screen first, then iterates through maze cells. Overlays the player '@' symbol on their current position. Other cells show their maze character ('#', '.', 'E', etc.).

Parameters

<i>maze</i>	Const reference to the maze grid.
<i>player</i>	Const reference to the player position.

Here is the call graph for this function:



Here is the caller graph for this function:

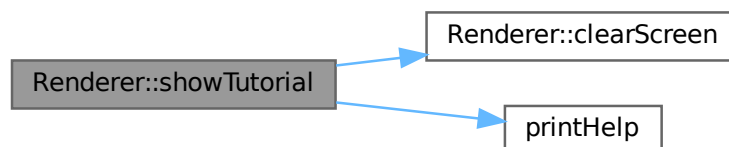


3.5.3.3 showTutorial()

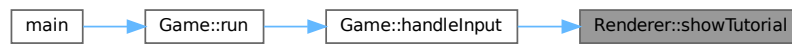
```
void Renderer::showTutorial ( ) const
```

Shows the game tutorial/help screen.

Clears screen, prints help text using `printHelp()`, then waits for Enter key press before returning. Here is the call graph for this function:



Here is the caller graph for this function:



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

- [renderer.hpp](#)
- [renderer.cpp](#)

Chapter 4

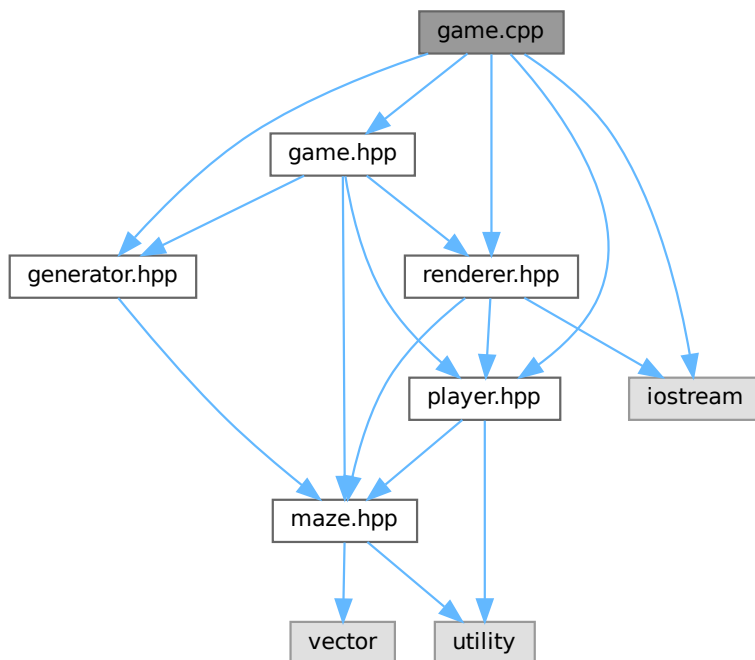
File Documentation

4.1 game.cpp File Reference

Implementation of the [Game](#) class and helper menu functions.

```
#include "game.hpp"  
#include "player.hpp"  
#include "renderer.hpp"  
#include "generator.hpp"  
#include <iostream>
```

Include dependency graph for game.cpp:



Functions

- char `helloFunction` ()
Displays the start menu and reads maze size preset choice.
- std::pair< int, int > `customChoice` ()
Asks the user for custom maze dimensions and validates them.

4.1.1 Detailed Description

Implementation of the `Game` class and helper menu functions.

Contains the main game loop, initialization logic (including maze size selection) and simple text-based user interface helpers.

4.1.2 Function Documentation

4.1.2.1 `customChoice()`

```
std::pair< int, int > customChoice ( )
```

Asks the user for custom maze dimensions and validates them.

The function prompts the user to enter width and height, checks that:

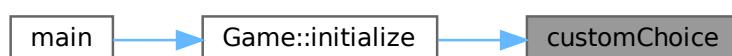
- both values are integers,
- both are at least 11,
- both are less than 100,
- the maze is not an extremely large square (to keep it playable), and then adjusts them to be odd numbers (required by the generator).

The input is requested repeatedly until valid dimensions are provided.

Returns

Pair (width, height) of validated, odd maze dimensions.

Here is the caller graph for this function:



4.1.2.2 helloFunction()

```
char helloFunction ( )
```

Displays the start menu and reads maze size preset choice.

Prints a welcome message and four options for maze size:

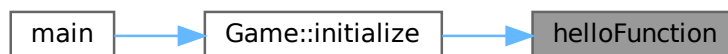
- 1: small preset,
- 2: medium preset,
- 3: large preset,
- 4: custom dimensions entered by the user.

The function keeps asking until the user enters a digit between '1' and '4'.

Returns

Character representing the chosen option ('1'-'4').

Here is the caller graph for this function:

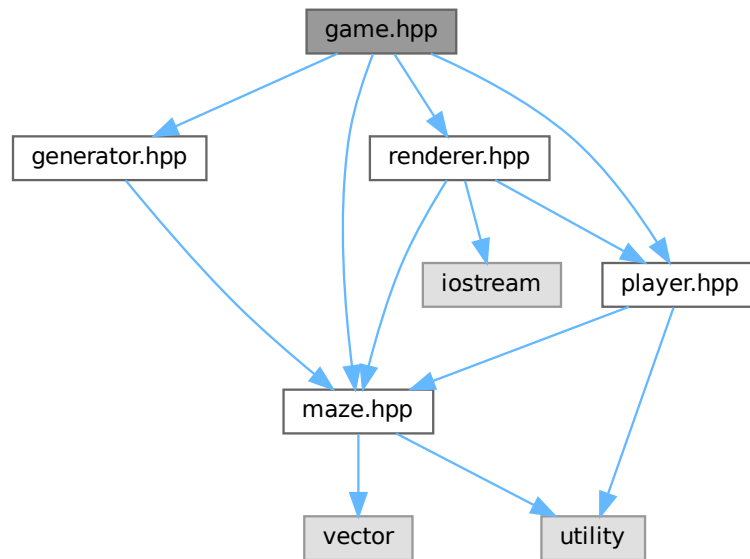


4.2 game.hpp File Reference

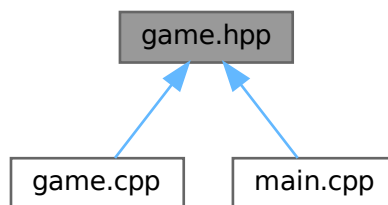
```
#include "generator.hpp"  
#include "maze.hpp"  
#include "player.hpp"
```

```
#include "renderer.hpp"
```

Include dependency graph for game.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Game](#)
High-level controller of the maze game.

4.3 game.hpp

[Go to the documentation of this file.](#)

00001 //

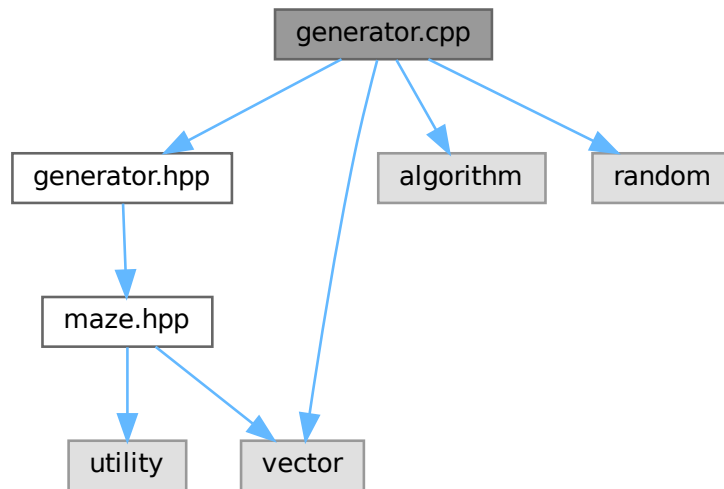
```
00002 // Created by antin on 24/12/2025.
00003 //
00004
00005 #ifndef CPPSEMESTRALPRJCT_GAME_HPP
00006 #define CPPSEMESTRALPRJCT_GAME_HPP
00007 #include "generator.hpp"
00008 #include "maze.hpp"
00009 #include "player.hpp"
00010 #include "renderer.hpp"
00011
00020 class Game {
00021     private:
00023         Maze maze;
00024
00026         Generator generator;
00027
00029         Player player;
00030
00032         Renderer renderer;
00033
00035         bool isRunning;
00036
00043         void handleInput();    //handles input from keyboard
00044
00051         void update();
00052
00059         void render() const;    //asks Renderer to draw an updated map
00060
00061     public:
00067         Game();
00068
00079         void initialize();
00080
00087         void run();
00088
00089 };
00090
00091 #endif //CPPSEMESTRALPRJCT_GAME_HPP
```

4.4 generator.cpp File Reference

Implementation of the maze generator based on recursive backtracking.

```
#include "generator.hpp"
#include "vector"
#include "algorithm"
#include "random"
```

Include dependency graph for generator.cpp:



4.4.1 Detailed Description

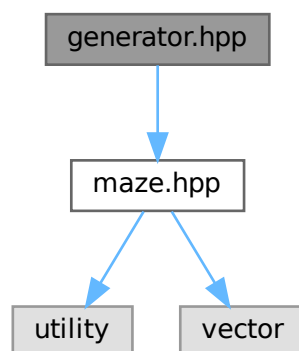
Implementation of the maze generator based on recursive backtracking.

This module provides the implementation of the [Generator](#) class which carves a random perfect maze by recursively visiting cells and removing walls between them.

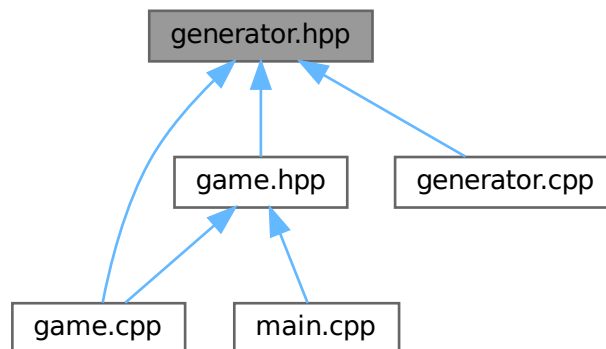
4.5 generator.hpp File Reference

```
#include "maze.hpp"
```

Include dependency graph for generator.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Generator](#)

Generates random mazes using the recursive backtracking algorithm.

4.6 generator.hpp

[Go to the documentation of this file.](#)

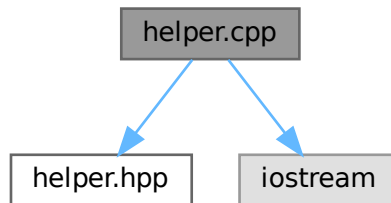
```

00001 //
00002 // Created by antin on 24/12/2025.
00003 //
00004
00005 #ifndef CPPSEMESTRALPRJCT_GENERATOR_HPP
00006 #define CPPSEMESTRALPRJCT_GENERATOR_HPP
00007 #include "maze.hpp"
00016 class Generator {
00017     private:
00030         void carvePassages(Maze& maze, int x, int y);
00031
00032     public:
00038         Generator();
00039
00051         void generate(Maze& maze, int startX, int startY);
00052
00053 };
00054
00055 #endif //CPPSEMESTRALPRJCT_GENERATOR_HPP
  
```

4.7 helper.cpp File Reference

Implementation of small helper utilities for the game.

```
#include "helper.hpp"
#include <iostream>
Include dependency graph for helper.cpp:
```



Functions

- void `printHelp()`
Prints a short help message to the standard output.

4.7.1 Detailed Description

Implementation of small helper utilities for the game.

Currently, contains only the `printHelp()` function that prints a brief description of controls and the game goal.

4.7.2 Function Documentation

4.7.2.1 `printHelp()`

```
void printHelp ( )
```

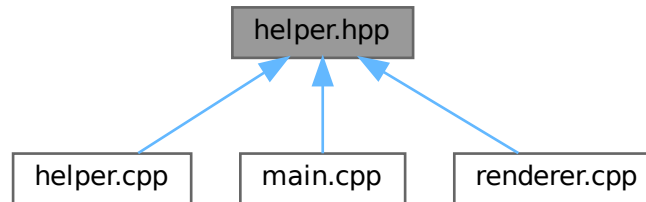
Prints a short help message to the standard output.

The help text explains basic controls, the meaning of the player symbol and the objective of the game. Intended to be called from the menu or on user request. Here is the caller graph for this function:



4.8 helper.hpp File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void `printHelp()`
Prints a short help message to the standard output.

4.8.1 Function Documentation

4.8.1.1 printHelp()

```
void printHelp ( )
```

Prints a short help message to the standard output.

The help text explains basic controls, the meaning of the player symbol and the objective of the game. Intended to be called from the menu or on user request. Here is the caller graph for this function:



4.9 helper.hpp

[Go to the documentation of this file.](#)

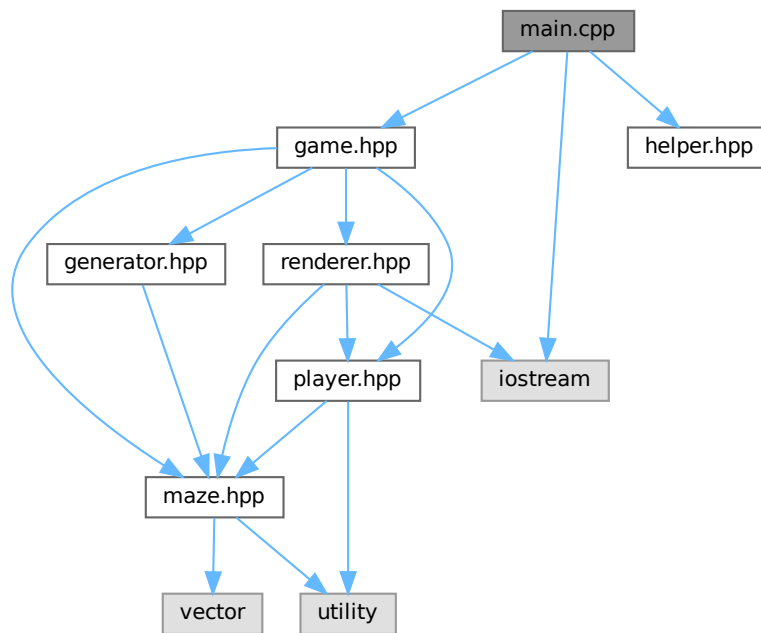
```

00001 //
00002 // Created by antin on 25/12/2025.
00003 //
00004
00005 #ifndef CPPSEMESTRALPRJCT_HELPER_HPP
00006 #define CPPSEMESTRALPRJCT_HELPER_HPP
00007
00015 void printHelp();
00016
00017
00018 #endif //CPPSEMESTRALPRJCT_HELPER_HPP
  
```

4.10 main.cpp File Reference

```
#include "game.hpp"
#include "helper.hpp"
#include <iostream>
```

Include dependency graph for main.cpp:



Functions

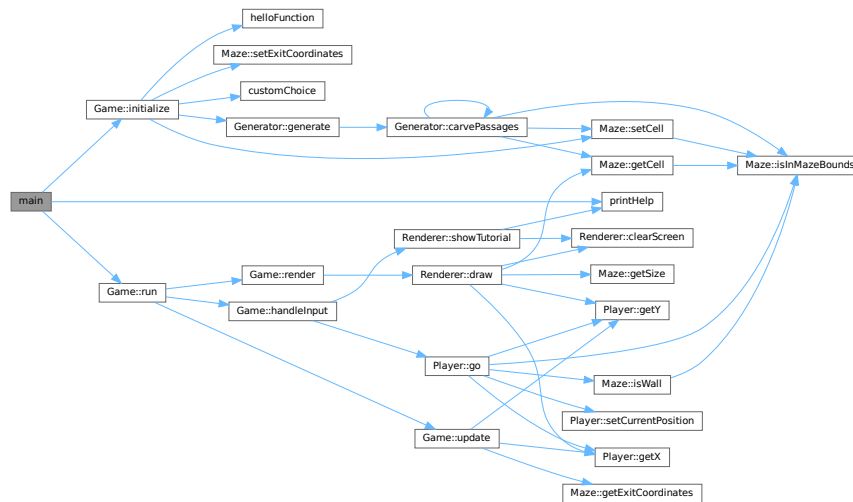
- `int main (int argc, char *argv[])`

4.10.1 Function Documentation

4.10.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Here is the call graph for this function:



4.11 maze.cpp File Reference

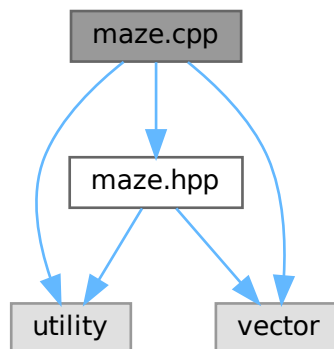
Implementation of the **Maze** class.

```
#include "maze.hpp"
```

```
#include <utility>
```

```
#include <vector>
```

Include dependency graph for maze.cpp:



4.11.1 Detailed Description

Implementation of the **Maze** class.

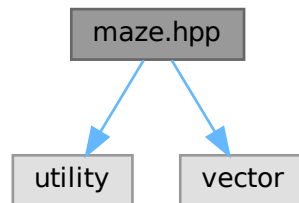
Provides constructors and member functions to manipulate the internal 2D grid of the maze, including size management, cell access and boundary checks.

4.12 maze.hpp File Reference

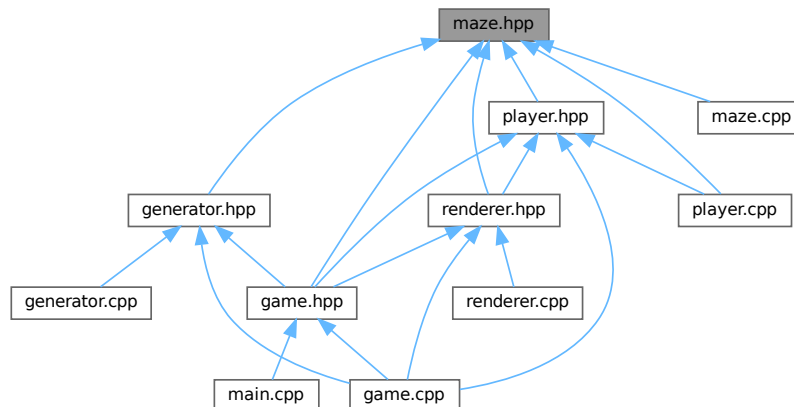
```
#include <utility>
```

```
#include <vector>
```

Include dependency graph for maze.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Maze](#)

Represents a maze structure.

4.13 maze.hpp

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by antin on 23/12/2025.
00003 //
00004
```

```

00005 #ifndef CPPSEMESTRALPRJCT_MAZE_HPP
00006 #define CPPSEMESTRALPRJCT_MAZE_HPP
00007 #include <utility>
00008 #include <vector>
00009
00017 class Maze {
00018     private:
00019         std::pair<int, int> size;
00020         std::vector<std::vector<char>> maze;
00021         std::pair<int, int> exitCoordinates;
00022     public:
00023         //constructors
00024         Maze();
00025
00031         Maze(int width, int height);
00032         //getters
00037         std::pair<int, int> getSize() const;
00038
00043         std::pair<int, int> getExitCoordinates() const;
00044
00051         char getCell(int x, int y) const;
00052         //setters
00053         void setSize(int width, int height);
00054         void setExitCoordinates(int x, int y);
00055
00062         void setCell(int x, int y, char value);
00063         //functions
00070         bool isInMazeBounds(int x, int y) const;
00077         bool isWall(int x, int y) const;
00078 };
00079
00080 #endif //CPPSEMESTRALPRJCT_MAZE_HPP

```

4.14 player.cpp File Reference

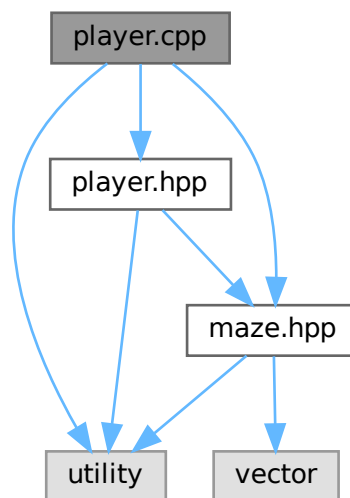
Implementation of the [Player](#) class.

```

#include "player.hpp"
#include "maze.hpp"
#include <utility>

```

Include dependency graph for player.cpp:



4.14.1 Detailed Description

Implementation of the [Player](#) class.

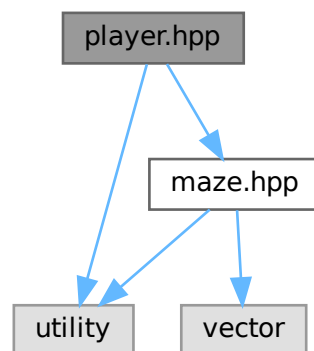
Provides constructors and methods for querying and updating the player's position, including movement constrained by the [Maze](#).

4.15 player.hpp File Reference

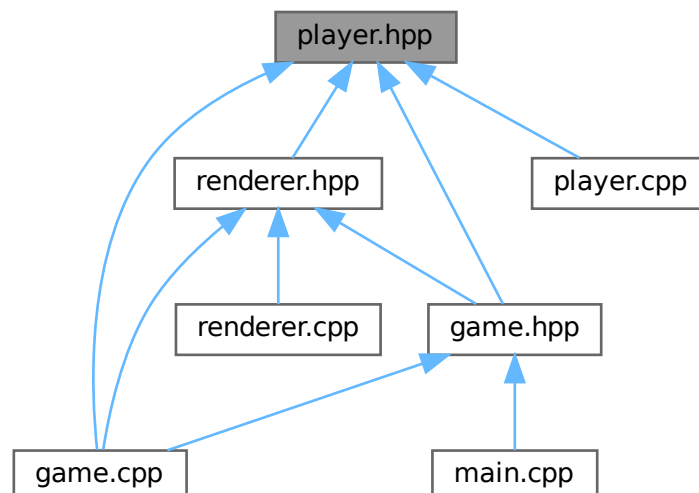
```
#include <utility>
```

```
#include "maze.hpp"
```

Include dependency graph for player.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Player](#)

Represents the player in the maze.

4.16 player.hpp

[Go to the documentation of this file.](#)

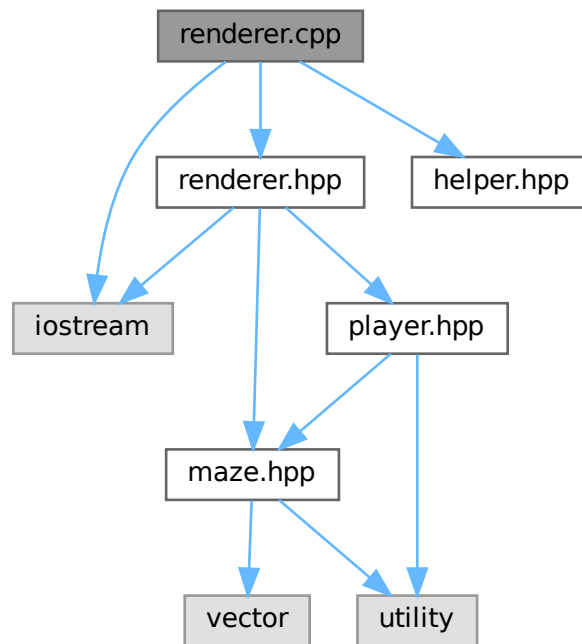
```
00001 //
00002 // Created by antin on 23/12/2025.
00003 //
00004
00005 #ifndef CPPSEMESTRALPRJCT_PLAYER_HPP
00006 #define CPPSEMESTRALPRJCT_PLAYER_HPP
00007
00008 #include <utility>
00009 #include "maze.hpp"
00010
00011 class Player {
00012 private:
00013     std::pair<int,int> currentPosition;
00014
00015 public:
00016     // constructors
00017     Player();
00018     Player(int x, int y);
00019
00020     // getters
00021     int getX() const;
00022     int getY() const;
00023     std::pair<int,int> getCurrentPosition() const;
00024
00025     // setters
00026     void setCurrentPosition(int x, int y);
00027     void go(char direction, const Maze& maze);
00028 };
00029 #endif //CPPSEMESTRALPRJCT_PLAYER_HPP
```

4.17 renderer.cpp File Reference

Implementation of the [Renderer](#) class.

```
#include "renderer.hpp"
#include "helper.hpp"
#include <iostream>
```

Include dependency graph for `renderer.cpp`:



4.17.1 Detailed Description

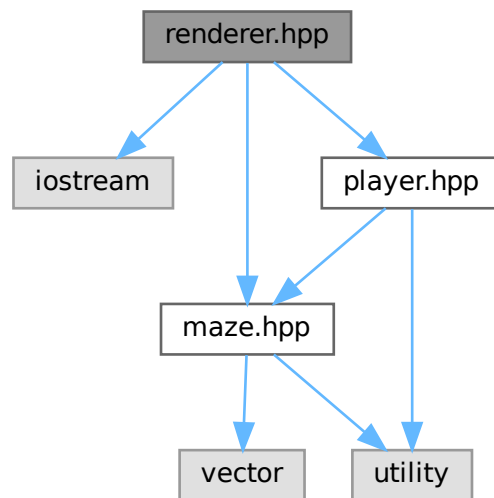
Implementation of the [Renderer](#) class.

Renders the maze game state to console using ANSI sequences for screen clearing and efficient text output.

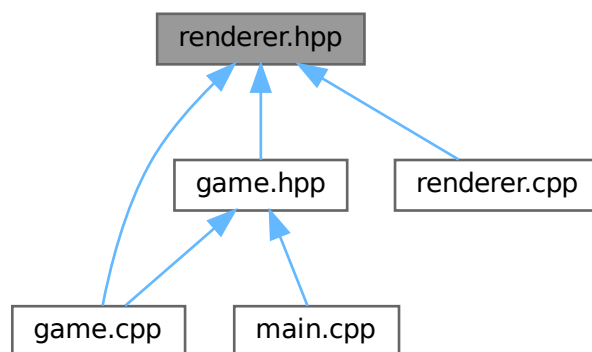
4.18 `renderer.hpp` File Reference

```
#include <iostream>
#include "maze.hpp"
#include "player.hpp"
```

Include dependency graph for renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [Renderer](#)
Handles console output for the maze game.

Macros

- `#define` [ANSI_CLEAR](#) `"\x1B[2J\x1B[H"`

4.18.1 Macro Definition Documentation

4.18.1.1 ANSI_CLEAR

```
#define ANSI_CLEAR "\x1B[2J\x1B[H"
```

4.19 renderer.hpp

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by antin on 24/12/2025.
00003 //
00004
00005 #ifndef CPPSEMESTRALPRJCT_RENDERER_HPP
00006 #define CPPSEMESTRALPRJCT_RENDERER_HPP
00007
00008 #define ANSI_CLEAR "\x1B[2J\x1B[H"
00009
00010 #include <iostream>
00011 #include "maze.hpp"
00012 #include "player.hpp"
00013
00021 class Renderer {
00022 private:
00029     void clearScreen() const {
00030         std::cout << ANSI_CLEAR << std::flush;
00031     }
00032
00033 public:
00037     Renderer();
00038
00049     void draw(const Maze &maze, const Player &player) const;
00050
00057     void showTutorial() const;
00058 };
00059
00060 #endif //CPPSEMESTRALPRJCT_RENDERER_HPP
```

Index

ANSI_CLEAR
 renderer.hpp, [46](#)

carvePassages
 Generator, [12](#)

clearScreen
 Renderer, [25](#)

currentPosition
 Player, [24](#)

customChoice
 game.cpp, [30](#)

draw
 Renderer, [25](#)

exitCoordinates
 Maze, [19](#)

Game, [5](#)
 Game, [6](#)
 generator, [10](#)
 handleInput, [7](#)
 initialize, [7](#)
 isRunning, [10](#)
 maze, [10](#)
 player, [11](#)
 render, [8](#)
 renderer, [11](#)
 run, [9](#)
 update, [9](#)
game.cpp, [29](#)
 customChoice, [30](#)
 helloFunction, [30](#)
game.hpp, [31](#)
generate
 Generator, [13](#)
Generator, [11](#)
 carvePassages, [12](#)
 generate, [13](#)
 Generator, [12](#)
generator
 Game, [10](#)
generator.cpp, [33](#)
generator.hpp, [34](#)
getCell
 Maze, [15](#)
getCurrentPosition
 Player, [21](#)
getExitCoordinates
 Maze, [16](#)

getSize
 Maze, [16](#)
getX
 Player, [21](#)
getY
 Player, [22](#)
go
 Player, [22](#)

handleInput
 Game, [7](#)
helloFunction
 game.cpp, [30](#)
helper.cpp, [35](#)
 printHelp, [36](#)
helper.hpp, [37](#)
 printHelp, [37](#)

initialize
 Game, [7](#)
isInMazeBounds
 Maze, [17](#)
isRunning
 Game, [10](#)
isWall
 Maze, [17](#)

main
 main.cpp, [38](#)
main.cpp, [38](#)
 main, [38](#)
Maze, [14](#)
 exitCoordinates, [19](#)
 getCell, [15](#)
 getExitCoordinates, [16](#)
 getSize, [16](#)
 isInMazeBounds, [17](#)
 isWall, [17](#)
 Maze, [15](#)
 maze, [19](#)
 setCell, [18](#)
 setExitCoordinates, [19](#)
 setSize, [19](#)
 size, [20](#)
maze
 Game, [10](#)
 Maze, [19](#)
maze.cpp, [39](#)
maze.hpp, [40](#)

Player, [20](#)

- currentPosition, [24](#)
- getCurrentPosition, [21](#)
- getX, [21](#)
- getY, [22](#)
- go, [22](#)
- Player, [21](#)
- setCurrentPosition, [23](#)
- player
 - Game, [11](#)
- player.cpp, [41](#)
- player.hpp, [42](#)
- printHelp
 - helper.cpp, [36](#)
 - helper.hpp, [37](#)
- render
 - Game, [8](#)
- Renderer, [24](#)
 - clearScreen, [25](#)
 - draw, [25](#)
 - Renderer, [25](#)
 - showTutorial, [26](#)
- renderer
 - Game, [11](#)
- renderer.cpp, [43](#)
- renderer.hpp, [44](#)
 - ANSI_CLEAR, [46](#)
- run
 - Game, [9](#)
- setCell
 - Maze, [18](#)
- setCurrentPosition
 - Player, [23](#)
- setExitCoordinates
 - Maze, [19](#)
- setSize
 - Maze, [19](#)
- showTutorial
 - Renderer, [26](#)
- size
 - Maze, [20](#)
- update
 - Game, [9](#)