Maze Game

1.0

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

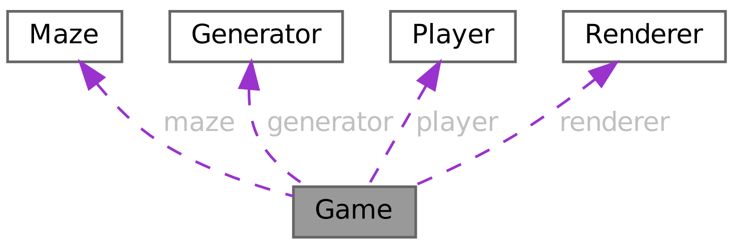
Class Documentation

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().

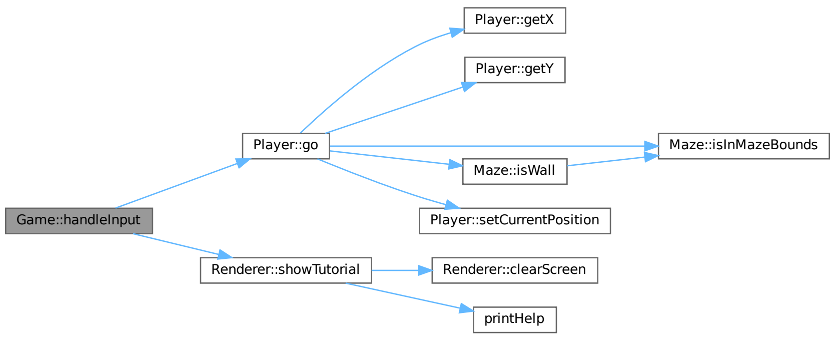
3.1.3 Member Function Documentation

3.1.3.1 handleInput()

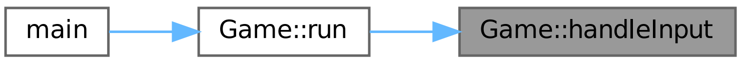
```
void Game::handleInput ( )  [private]
```

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()

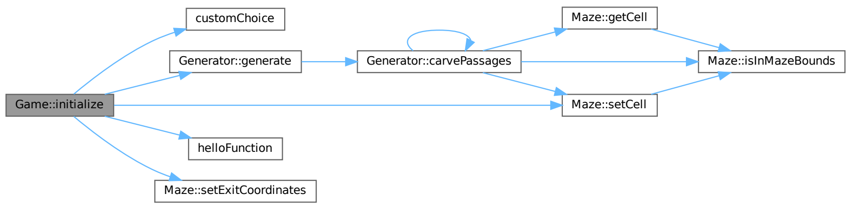
```
void Game::initialize ( )
```

Initializes the game before the main loop starts.

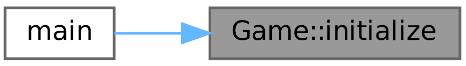
Shows a menu that lets the user choose a predefined maze size or enter a custom size, then:

- 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:

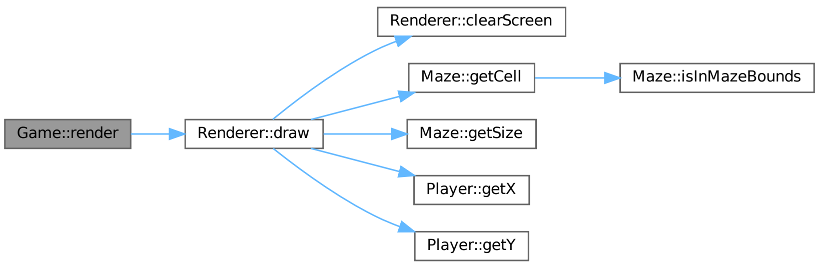


3.1.3.3 render()

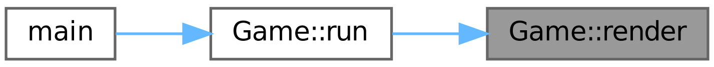
```
void Game::render ( ) const  [private]
```

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:

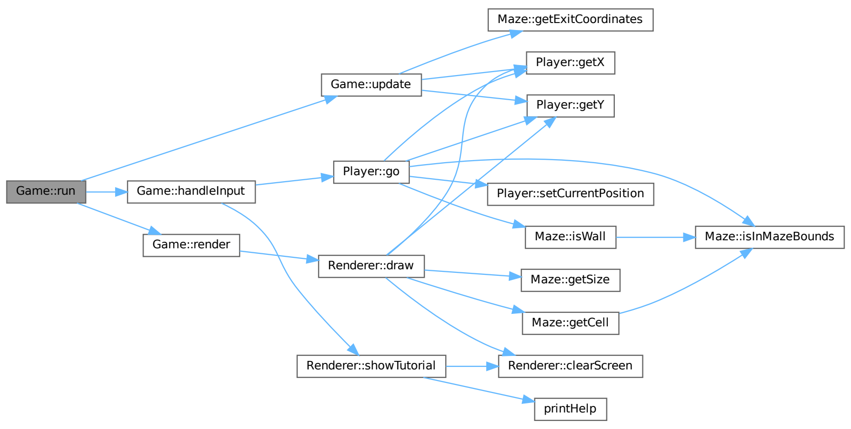


3.1.3.4 run()

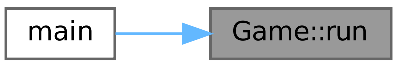
```
void Game::run ( )
```

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:

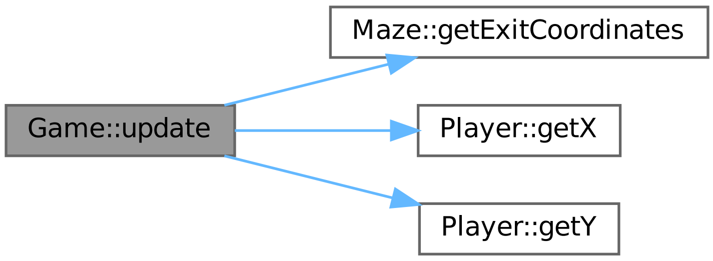


3.1.3.5 update()

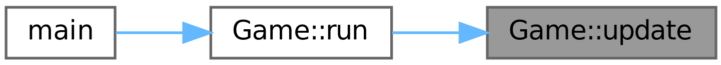
```
void Game::update ( )  [private]
```

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

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

The algorithm:

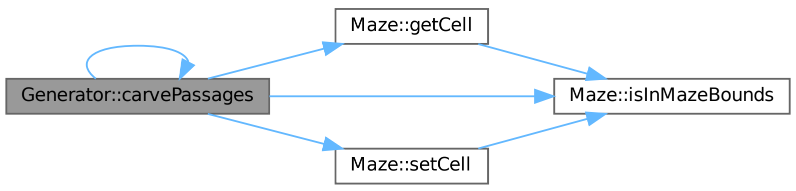
- marks the current cell as a passage ('.'),
- creates a list of four possible directions (up, down, left, right),
- shuffles the directions randomly,
- for each direction computes the neighbour two cells away,
- if the neighbour is inside the maze and still a wall ('#'), removes the wall between current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

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

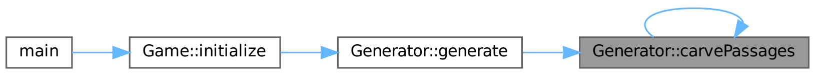
Parameters

maze	Reference to the maze being modified.
x	Current cell x-coordinate.
y	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 )
```

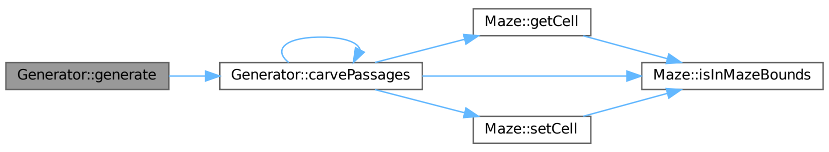
Generates a maze starting from the given cell.

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.

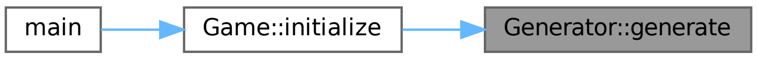
Parameters

maze	Reference to the maze that will be filled with passages.
startX	Starting cell x-coordinate (should be odd).
startY	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 dimesnions.

Parameters

width	The width of the maze
height	The height of the maze

3.3.3 Member Function Documentation

3.3.3.1 getCell()

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

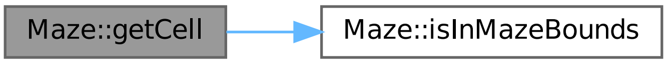
Gets a cell value at the specified position.

Parameters

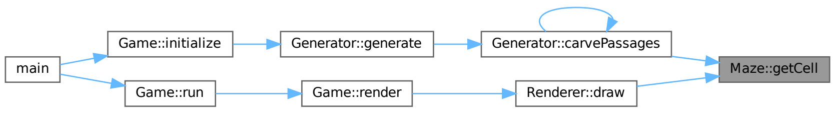
x	X-coordinate
y	Y-coordinate

Returns

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.2 getExitCoordinates()

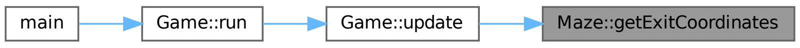
```
std::pair< int, int > Maze::getExitCoordinates ( ) const
```

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()

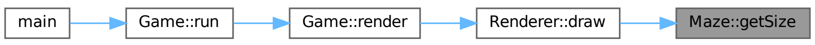
```
std::pair< int, int > Maze::getSize ( ) const
```

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.

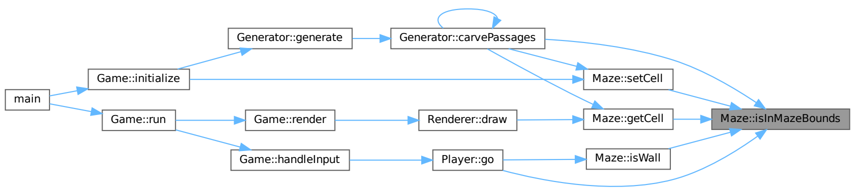
Parameters

x	X-coordinate to check
y	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.

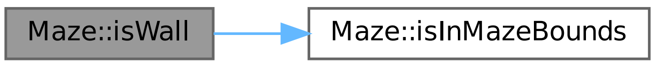
Parameters

x	X-coordinate
y	Y-coordinate

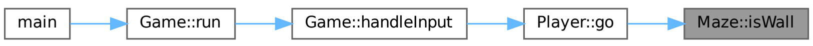
Returns

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()

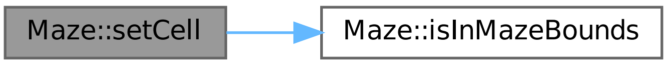
```
void Maze::setCell (
             int x,
             int y,
             char value )
```

Sets a cell value at the specified position.

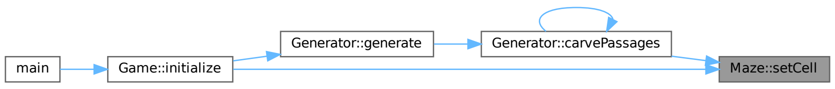
Parameters

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

Here is the call graph for this function:



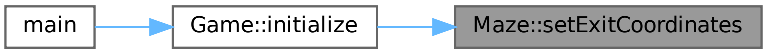
Here is the caller graph for this function:



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

x	Initial x-coordinate.
y	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()

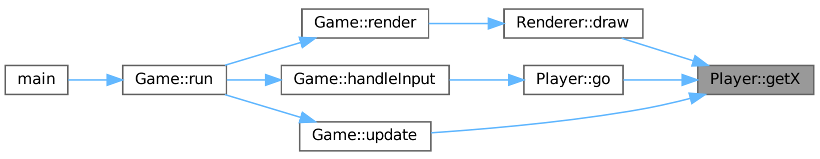
```
int Player::getX ( ) const
```

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()

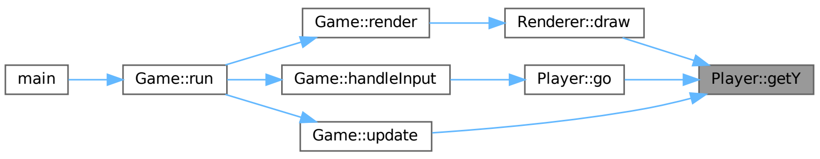
```
int Player::getY ( ) const
```

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:

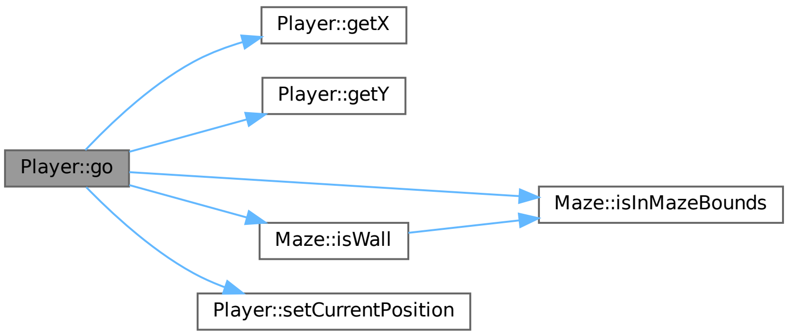
- 'W'/'w' – up,
- 'S'/'s' – down,
- 'A'/'a' – left,
- 'D'/'d' – right.

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

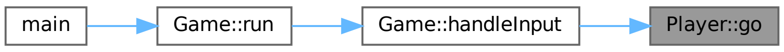
Parameters

direction	Character representing the move command.
maze	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()

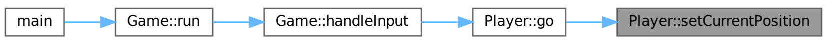
```
void Player::setCurrentPosition (
             int x,
             int y )
```

Sets the current position of the player.

Parameters

x	New x-coordinate.
y	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.

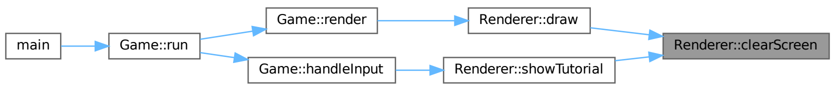
3.5.3 Member Function Documentation

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
```

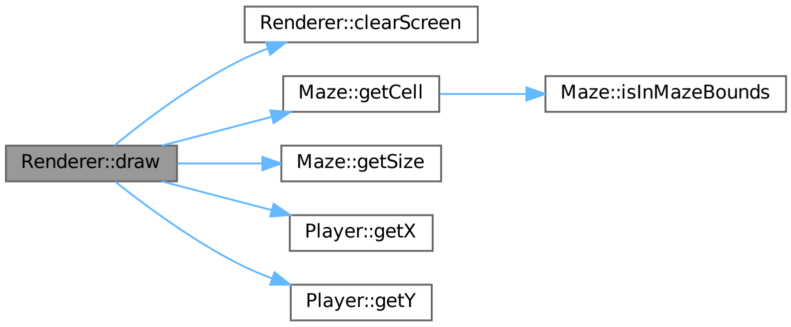
Draws the complete maze with player position.

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

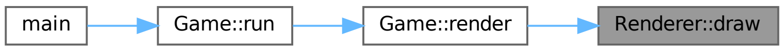
Parameters

maze	Const reference to the maze grid.
player	Const reference to the player position.

Here is the call graph for this function:



Here is the caller graph for this function:

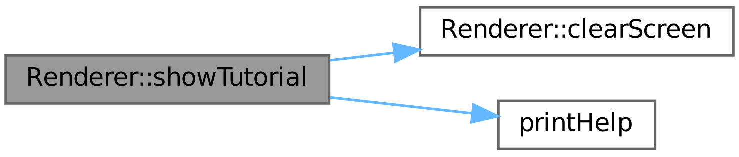


3.5.3.3 showTutorial()

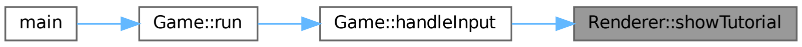
```
void Renderer::showTutorial ( ) const
```

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

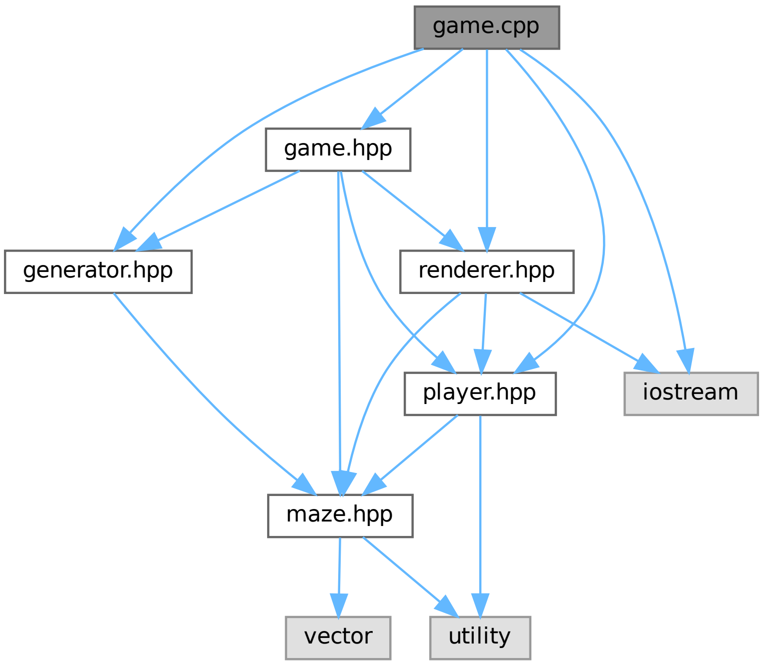
File Documentation

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:

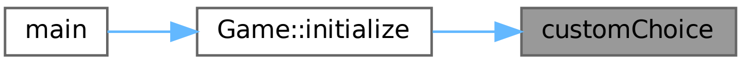
- both values are integers,
- both are at least 11,
- both are less than 100,
- the maze is not an extremely large square (to keep it playable), and then adjusts them to be odd numbers (required by the generator).

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:

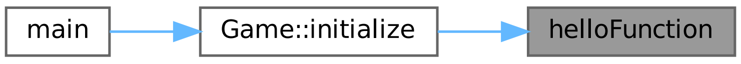
- 1: small preset,
- 2: medium preset,
- 3: large preset,
- 4: custom dimensions entered by the user.

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

Returns

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

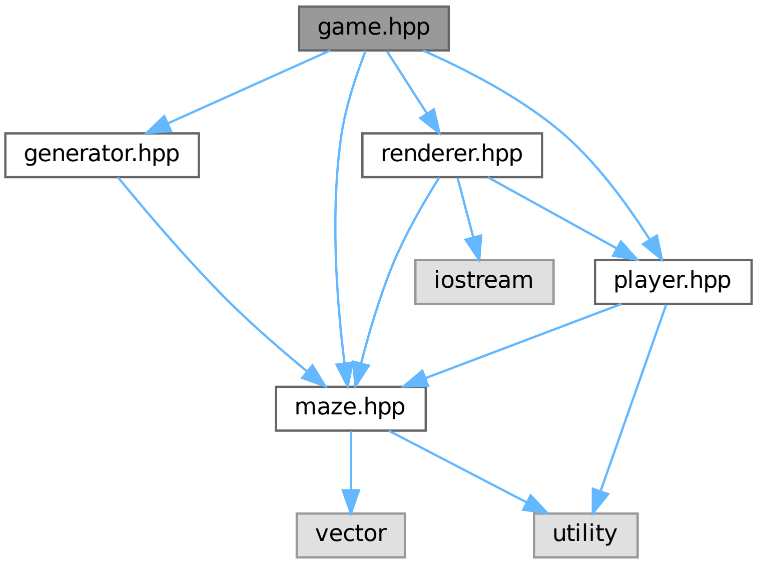
Here is the caller graph for this function:



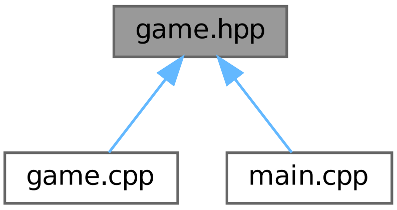
4.2 game.hpp File Reference

```
#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 //
```

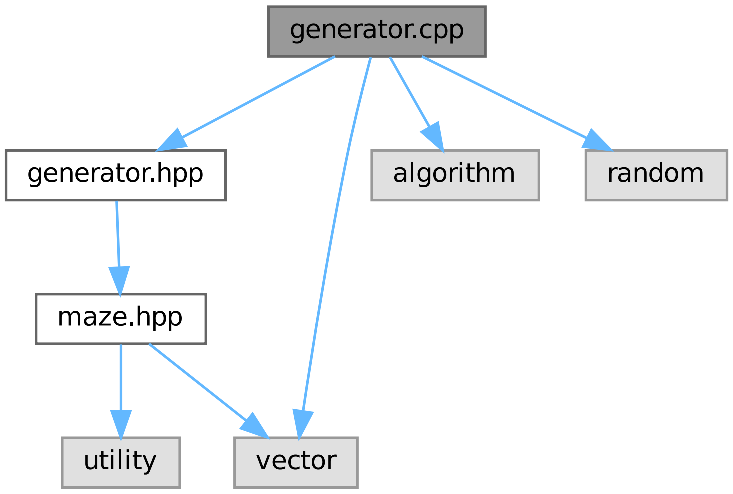
```
// Created by antin on 24/12/2025.
//

#ifndef CPPSEMESTRALPRJCT_GAME_HPP
#define CPPSEMESTRALPRJCT_GAME_HPP
#include "generator.hpp"
#include "maze.hpp"
#include "player.hpp"
#include "renderer.hpp"

class Game {
    private:
        Maze maze;

        Generator generator;

        Player player;

        Renderer renderer;

        bool isRunning;

        void handleInput();     //handles input from keyboard

        void update();

        void render() const;    //asks Renderer to draw an updated map

    public:
        Game();

        void initialize();

        void run();

};

#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

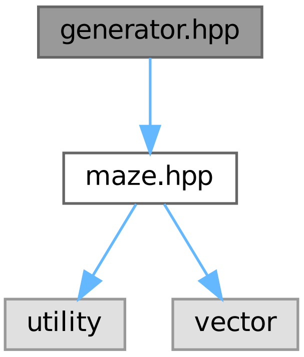
Implementation of the maze generator based on recursive backtracking.

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

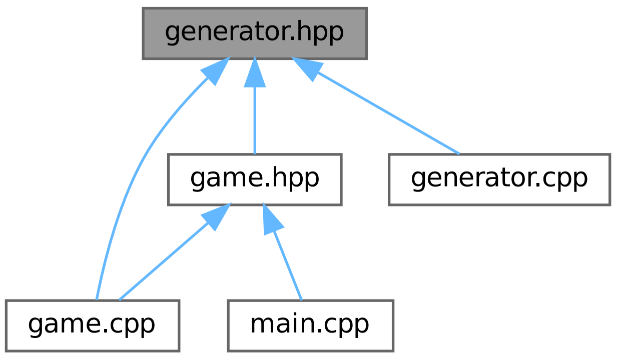
4.5 generator.hpp File Reference

```
#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.

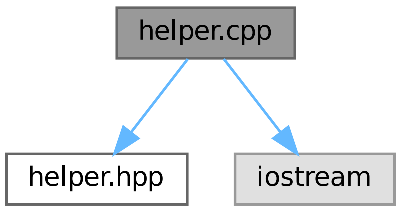
```
//
// Created by antin on 24/12/2025.
//

#ifndef CPPSEMESTRALPRJCT_GENERATOR_HPP
#define CPPSEMESTRALPRJCT_GENERATOR_HPP
#include "maze.hpp"
class Generator {
    private:
        void carvePassages(Maze& maze, int x, int y);

    public:
        Generator();

        void generate(Maze& maze, int startX, int startY);

};

#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.

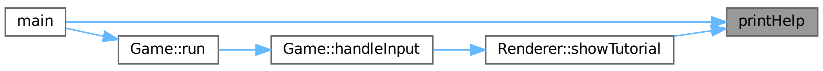
4.7.2 Function Documentation

4.7.2.1 printHelp()

```
void printHelp ( )
```

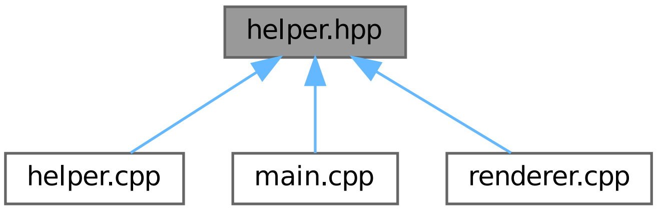
Prints a short help message to the standard output.

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.

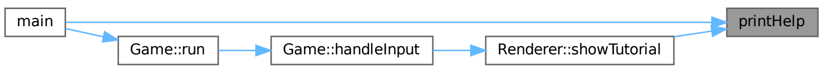
4.8.1 Function Documentation

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

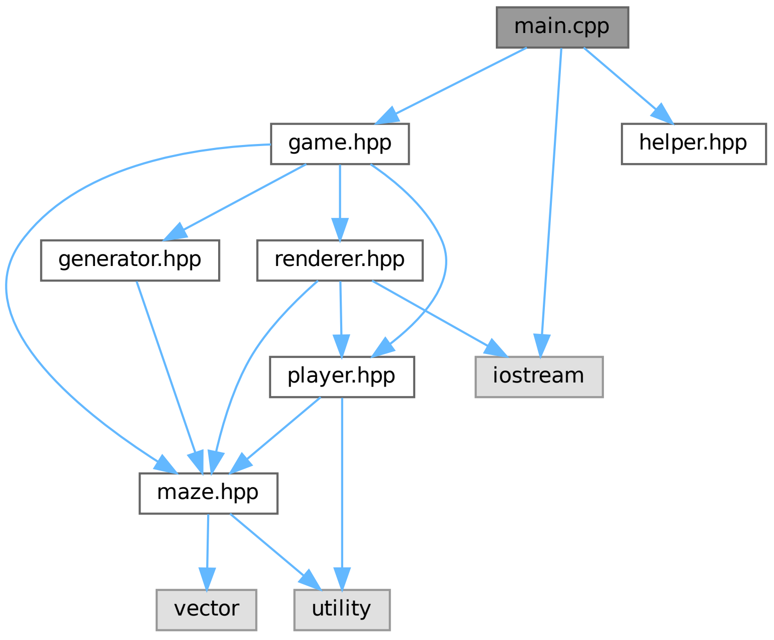
Go to the documentation of this file.

```
//
// Created by antin on 25/12/2025.
//

#ifndef CPPSEMESTRALPRJCT_HELPER_HPP
#define CPPSEMESTRALPRJCT_HELPER_HPP

void printHelp();


#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

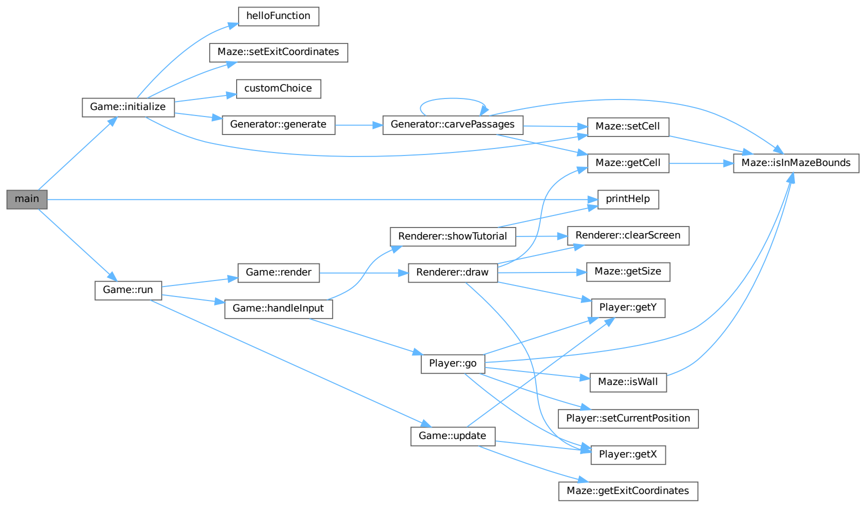
- int main (int argc, char ∗argv[ ])

4.10.1 Function Documentation

4.10.1.1 main()

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

Here is the call graph for this function:

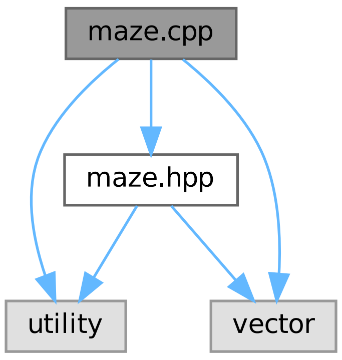


4.11 maze.cpp File Reference

Implementation of the Maze class.

```
#include "maze.hpp"
#include <utility>
#include <vector>
```

Include dependency graph for maze.cpp:



4.11.1 Detailed Description

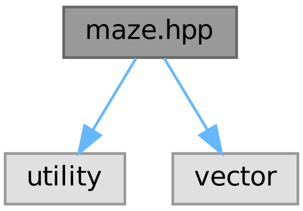
Implementation of the Maze class.

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

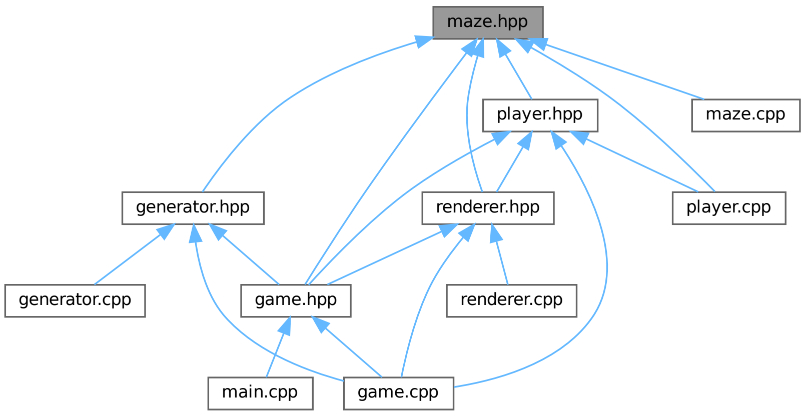
4.12 maze.hpp File Reference

```
#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.

```
//
// Created by antin on 23/12/2025.
//

```

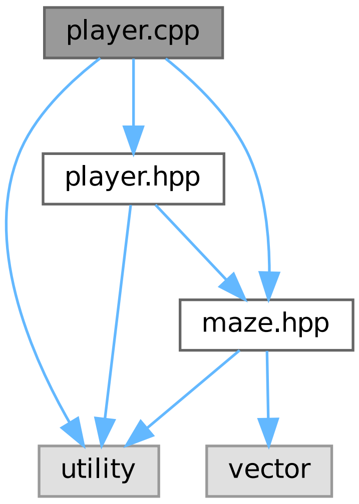
```
#ifndef CPPSEMESTRALPRJCT_MAZE_HPP
#define CPPSEMESTRALPRJCT_MAZE_HPP
#include <utility>
#include <vector>

class Maze {
    private:
        std::pair<int, int> size;
        std::vector<std::vector<char» maze;
        std::pair<int, int> exitCoordinates;
    public:
    //constructors
        Maze();

        Maze(int width, int height);
    //getters
        std::pair<int, int> getSize() const;

        std::pair<int, int> getExitCoordinates() const;

        char getCell(int x, int y) const;
    //setters
        void setSize(int width, int height);
        void setExitCoordinates(int x, int y);

        void setCell(int x, int y, char value);
    //functions
        bool isInMazeBounds(int x, int y) const;
        bool isWall(int x, int y) const;
};

#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

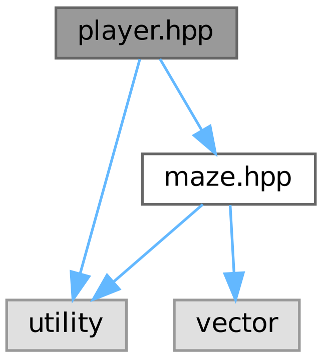
Implementation of the Player class.

Provides constructors and methods for querying and updating the player's position, including movement constrained by the Maze.

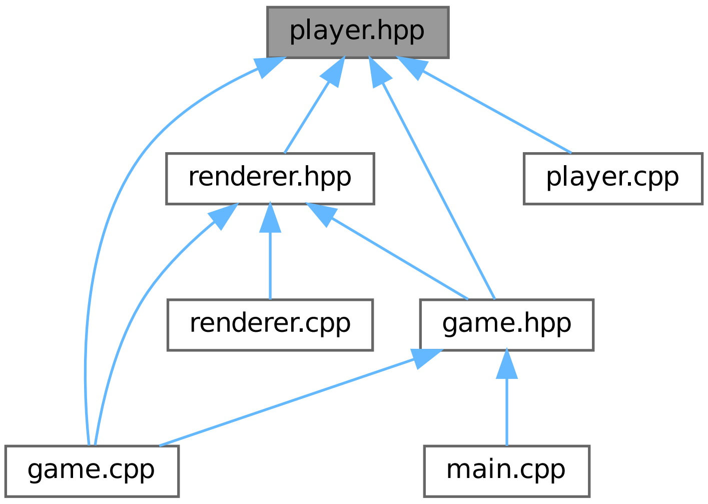
4.15 player.hpp File Reference

```
#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.

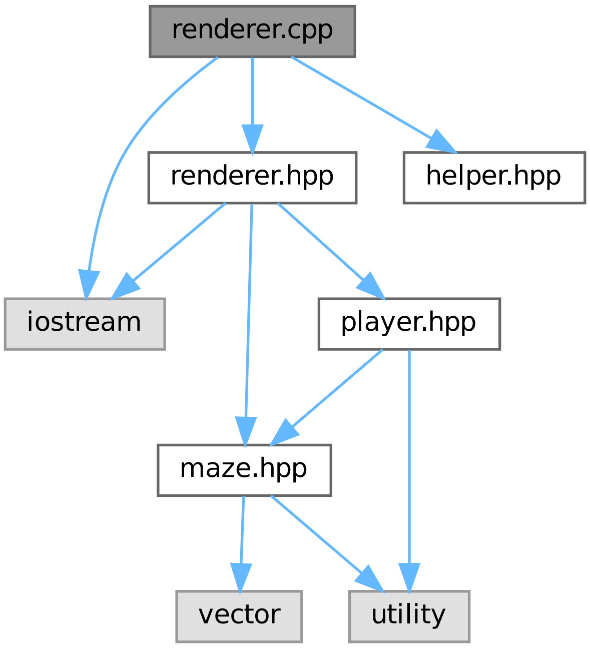
```
//
// Created by antin on 23/12/2025.
//

#ifndef CPPSEMESTRALPRJCT_PLAYER_HPP
#define CPPSEMESTRALPRJCT_PLAYER_HPP

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

class Player {
private:
    std::pair<int,int> currentPosition;

public:
    // constructors

    Player();

    Player(int x, int y);

    // getters

    int getX() const;

    int getY() const;

    std::pair<int,int> getCurrentPosition() const;

    // setters

    void setCurrentPosition(int x, int y);

    void go(char direction, const Maze& maze);
};

#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

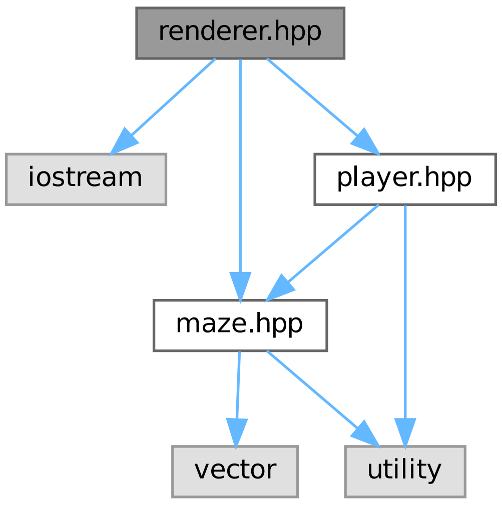
Implementation of the Renderer class.

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

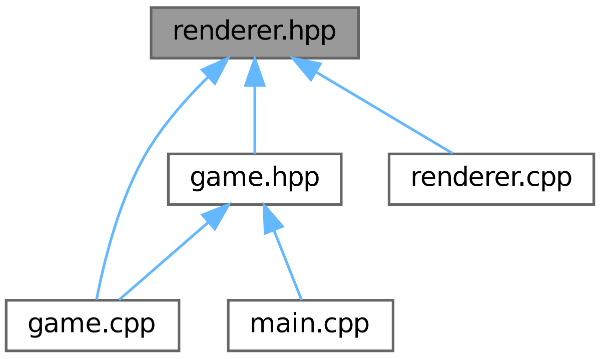
4.18 renderer.hpp File Reference

```
#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.

```
//
// Created by antin on 24/12/2025.
//

#ifndef CPPSEMESTRALPRJCT_RENDERER_HPP
#define CPPSEMESTRALPRJCT_RENDERER_HPP

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

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

class Renderer {
private:
    void clearScreen() const {
        std::cout « ANSI_CLEAR « std::flush;
    }

public:
    Renderer();

    void draw(const Maze &maze, const Player &player) const;

    void showTutorial() const;
};

#endif //CPPSEMESTRALPRJCT_RENDERER_HPP
```

Index