

SnakeGame

Generated by Doxygen 1.16.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 GameConfig Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 diff	5
3.1.2.2 height	6
3.1.2.3 speed	6
3.1.2.4 width	6
3.2 Node Struct Reference	6
3.2.1 Detailed Description	7
3.2.2 Constructor & Destructor Documentation	7
3.2.2.1 Node()	7
3.2.3 Member Data Documentation	7
3.2.3.1 next	7
3.2.3.2 prev	7
3.2.3.3 x	7
3.2.3.4 y	7
3.3 SnakeList Class Reference	8
3.3.1 Detailed Description	8
3.3.2 Constructor & Destructor Documentation	8
3.3.2.1 SnakeList() [1/3]	8
3.3.2.2 SnakeList() [2/3]	8
3.3.2.3 SnakeList() [3/3]	8
3.3.3 Member Function Documentation	9
3.3.3.1 isSnakeAt()	9
3.3.3.2 operator=() [1/2]	9
3.3.3.3 operator=() [2/2]	9
3.3.3.4 pop_back()	9
3.3.3.5 push_front()	10
3.3.3.6 sortContent()	10
4 File Documentation	11
4.1 Draw.cpp File Reference	11
4.1.1 Detailed Description	11
4.1.2 Function Documentation	12
4.1.2.1 Draw()	12
4.2 Draw.cpp	12

4.3 Draw.h File Reference	13
4.3.1 Detailed Description	13
4.3.2 Function Documentation	13
4.3.2.1 Draw()	13
4.4 Draw.h	14
4.5 GameConfig.cpp File Reference	14
4.5.1 Detailed Description	14
4.5.2 Function Documentation	15
4.5.2.1 loadConfig()	15
4.6 GameConfig.cpp	15
4.7 GameConfig.h File Reference	16
4.7.1 Detailed Description	16
4.7.2 Function Documentation	16
4.7.2.1 loadConfig()	16
4.8 GameConfig.h	17
4.9 main.cpp File Reference	17
4.9.1 Detailed Description	17
4.9.2 Function Documentation	17
4.9.2.1 main()	17
4.10 main.cpp	18
4.11 SaveHighScore.cpp File Reference	19
4.11.1 Detailed Description	19
4.11.2 Function Documentation	20
4.11.2.1 displayLeaderboard()	20
4.11.2.2 saveHighScore()	20
4.12 SaveHighScore.cpp	20
4.13 SaveHighScore.h File Reference	21
4.13.1 Detailed Description	21
4.13.2 Function Documentation	21
4.13.2.1 displayLeaderboard()	21
4.13.2.2 saveHighScore()	21
4.14 SaveHighScore.h	22
4.15 SnakeList.cpp File Reference	22
4.15.1 Detailed Description	22
4.16 SnakeList.cpp	22
4.17 SnakeList.h File Reference	23
4.17.1 Detailed Description	24
4.18 SnakeList.h	24
4.19 Types.h File Reference	25
4.19.1 Detailed Description	25
4.19.2 Enumeration Type Documentation	25
4.19.2.1 Difficulty	25

4.19.2.2 eDirection	25
4.20 Types.h	26
Index	27

Chapter 1

Class Index

1.1 Class List

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

GameConfig	Stores game parameters such as board dimensions and speed	5
Node	Represents a single segment of the snake's body	6
SnakeList	Doubly linked list for managing snake segments. Implements the Rule of Five and custom sorting/searching	8

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Draw.cpp	Implementation of the rendering engine	11
Draw.h	Functions responsible for rendering the game state to the console	13
GameConfig.cpp	Implementation of configuration loading	14
GameConfig.h	Configuration structures and functions for game settings	16
main.cpp	The main entry point for the Snake game	17
SaveHighScore.cpp	Implementation of the high score system	19
SaveHighScore.h	File I/O operations for saving and displaying player scores	21
SnakeList.cpp	Implementation of the SnakeList class and its memory management	22
SnakeList.h	Header file containing the Doubly Linked List implementation for the Snake game	23
Types.h	Common enumerations used across the project	25

Chapter 3

Class Documentation

3.1 GameConfig Struct Reference

Stores game parameters such as board dimensions and speed.

```
#include <GameConfig.h>
```

Public Attributes

- int [width](#)
Width of the game board.
- int [height](#)
Height of the game board.
- int [speed](#)
Delay between frames in milliseconds.
- [Difficulty](#) [diff](#)
Game difficulty level.

3.1.1 Detailed Description

Stores game parameters such as board dimensions and speed.

Definition at line [15](#) of file [GameConfig.h](#).

3.1.2 Member Data Documentation

3.1.2.1 diff

[Difficulty](#) [GameConfig::diff](#)

Game difficulty level.

Definition at line [19](#) of file [GameConfig.h](#).

3.1.2.2 height

```
int GameConfig::height
```

Height of the game board.

Definition at line 17 of file [GameConfig.h](#).

3.1.2.3 speed

```
int GameConfig::speed
```

Delay between frames in milliseconds.

Definition at line 18 of file [GameConfig.h](#).

3.1.2.4 width

```
int GameConfig::width
```

Width of the game board.

Definition at line 16 of file [GameConfig.h](#).

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

- [GameConfig.h](#)

3.2 Node Struct Reference

Represents a single segment of the snake's body.

```
#include <SnakeList.h>
```

Public Member Functions

- [Node](#) (int _x, int _y)

Public Attributes

- int [x](#)
- int [y](#)
Coordinates of the segment.
- std::unique_ptr< [Node](#) > [next](#)
Smart pointer to the next segment (forward ownership).
- [Node](#) * [prev](#)
Raw pointer to the previous segment (bidirectional link).

3.2.1 Detailed Description

Represents a single segment of the snake's body.

Definition at line 15 of file [SnakeList.h](#).

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

```
Node::Node (
    int _x,
    int _y)
```

Definition at line 9 of file [SnakeList.cpp](#).

3.2.3 Member Data Documentation

3.2.3.1 next

```
std::unique_ptr<Node> Node::next
```

Smart pointer to the next segment (forward ownership).

Definition at line 17 of file [SnakeList.h](#).

3.2.3.2 prev

```
Node* Node::prev
```

Raw pointer to the previous segment (bidirectional link).

Definition at line 18 of file [SnakeList.h](#).

3.2.3.3 x

```
int Node::x
```

Definition at line 16 of file [SnakeList.h](#).

3.2.3.4 y

```
int Node::y
```

Coordinates of the segment.

Definition at line 16 of file [SnakeList.h](#).

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

- [SnakeList.h](#)
- [SnakeList.cpp](#)

3.3 SnakeList Class Reference

Doubly linked list for managing snake segments. Implements the Rule of Five and custom sorting/searching.

```
#include <SnakeList.h>
```

Public Member Functions

- [SnakeList](#) ()
- void [push_front](#) (int x, int y)
Adds a new segment to the front of the snake.
- void [pop_back](#) ()
Removes the last segment (tail) of the snake.
- bool [isSnakeAt](#) (int x, int y) const
Checks if a snake segment exists at given coordinates.
- [SnakeList](#) (const [SnakeList](#) &other)
- [SnakeList](#) ([SnakeList](#) &&other) noexcept
- [SnakeList](#) & operator= (const [SnakeList](#) &other)
- [SnakeList](#) & operator= ([SnakeList](#) &&other) noexcept
- void [sortContent](#) ()
Sorts segments by X coordinate using Bubble Sort.

3.3.1 Detailed Description

Doubly linked list for managing snake segments. Implements the Rule of Five and custom sorting/searching.

Definition at line 28 of file [SnakeList.h](#).

3.3.2 Constructor & Destructor Documentation

3.3.2.1 [SnakeList\(\)](#) [1/3]

```
SnakeList::SnakeList ()
```

Definition at line 11 of file [SnakeList.cpp](#).

3.3.2.2 [SnakeList\(\)](#) [2/3]

```
SnakeList::SnakeList (  
    const SnakeList & other)
```

Definition at line 54 of file [SnakeList.cpp](#).

3.3.2.3 [SnakeList\(\)](#) [3/3]

```
SnakeList::SnakeList (  
    SnakeList && other) [noexcept]
```

Definition at line 64 of file [SnakeList.cpp](#).

3.3.3 Member Function Documentation

3.3.3.1 isSnakeAt()

```
bool SnakeList::isSnakeAt (  
    int x,  
    int y) const
```

Checks if a snake segment exists at given coordinates.

Returns

true if collision detected.

Definition at line 42 of file [SnakeList.cpp](#).

3.3.3.2 operator=() [1/2]

```
SnakeList & SnakeList::operator= (  
    const SnakeList & other)
```

Definition at line 69 of file [SnakeList.cpp](#).

3.3.3.3 operator=() [2/2]

```
SnakeList & SnakeList::operator= (  
    SnakeList && other) [noexcept]
```

Definition at line 87 of file [SnakeList.cpp](#).

3.3.3.4 pop_back()

```
void SnakeList::pop_back ()
```

Removes the last segment (tail) of the snake.

Definition at line 27 of file [SnakeList.cpp](#).

3.3.3.5 push_front()

```
void SnakeList::push_front (
    int x,
    int y)
```

Adds a new segment to the front of the snake.

Parameters

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

Definition at line 12 of file [SnakeList.cpp](#).

3.3.3.6 sortContent()

```
void SnakeList::sortContent ()
```

Sorts segments by X coordinate using Bubble Sort.

Definition at line 97 of file [SnakeList.cpp](#).

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

- [SnakeList.h](#)
- [SnakeList.cpp](#)

Chapter 4

File Documentation

4.1 Draw.cpp File Reference

Implementation of the rendering engine.

```
#include "Draw.h"
```

Functions

- void [Draw](#) (const [GameConfig](#) &cfg, [SnakeList](#) &snake, int fruitX, int fruitY, int score)
Renders the game board, snake, and fruit.

4.1.1 Detailed Description

Implementation of the rendering engine.

- Uses Windows console API to position the cursor and draw the game board, snake segments, and fruits without flickering.

Definition in file [Draw.cpp](#).

4.1.2 Function Documentation

4.1.2.1 Draw()

```
void Draw (
    const GameConfig & cfg,
    SnakeList & snake,
    int fruitX,
    int fruitY,
    int score)
```

Renders the game board, snake, and fruit.

Parameters

<i>cfg</i>	Current game configuration.
<i>snake</i>	Reference to the SnakeList object.
<i>fruitX</i>	X-coordinate of the fruit.
<i>fruitY</i>	Y-coordinate of the fruit.
<i>score</i>	Current player score.

Definition at line 9 of file [Draw.cpp](#).

4.2 Draw.cpp

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

```
00001
00007 #include "Draw.h"
00008
00009 void Draw(const GameConfig& cfg, SnakeList& snake, int fruitX, int fruitY, int score) {
00010
00011     COORD cursorPosition;
00012     cursorPosition.X = 0;
00013     cursorPosition.Y = 0;
00014
00015     SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), cursorPosition);
00016     //upper wall
00017     for (int i = 0; i < (cfg.width/2)-8; i++) { std::cout << " "; }
00018     std::cout << "----SNAKE GAME---\n";
00019     for (int i = 0; i < cfg.width + 2; i++) { std::cout << "#"; }
00020     std::cout << "\n";
00021
00022
00023     //left wall->checking if snake is on j,i position-> right wall
00024     for (int i = 0; i < cfg.height; i++) {
00025         for (int j = 0; j < cfg.width; j++) {
00026             if (j == 0) std::cout << "#";
00027             if (snake.isSnakeAt(j, i)) {
00028                 std::cout << "O";
00029             }
00030             else if (j == fruitX && i == fruitY)
00031             {
00032                 std::cout << "Q";
00033             }
00034             else {
00035                 std::cout << " ";
00036             }
00037             if (j == cfg.width - 1) std::cout << "#";
00038         }
00039         std::cout << "\n";
00040     }
00041     //bottom wall
00042     for (int i = 0; i < cfg.width + 2; i++) {
00043         std::cout << "#";
00044     }
```

```

00045     std::cout << "\n";
00046
00047     std::cout << "-----";
00048
00049     std::cout << "\n";
00050     //Final score= Points * speed!
00051     std::cout << "Points: " << score << "| map config: " << cfg.width << "x" << cfg.height << "| speed : " <<
110-cfg.speed << "\n";
00052
00053
00054
00055 }
```

4.3 Draw.h File Reference

Functions responsible for rendering the game state to the console.

```

#include "GameConfig.h"
#include "SnakeList.h"
```

Functions

- void [Draw](#) (const [GameConfig](#) &cfg, [SnakeList](#) &snake, int fruitX, int fruitY, int score)
Renders the game board, snake, and fruit.

4.3.1 Detailed Description

Functions responsible for rendering the game state to the console.

Definition in file [Draw.h](#).

4.3.2 Function Documentation

4.3.2.1 Draw()

```

void Draw (
    const GameConfig & cfg,
    SnakeList & snake,
    int fruitX,
    int fruitY,
    int score)
```

Renders the game board, snake, and fruit.

Parameters

<i>cfg</i>	Current game configuration.
<i>snake</i>	Reference to the SnakeList object.
<i>fruitX</i>	X-coordinate of the fruit.
<i>fruitY</i>	Y-coordinate of the fruit.

<code>score</code>	Current player score.
--------------------	-----------------------

Definition at line 9 of file [Draw.cpp](#).

4.4 Draw.h

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

```
00001
00005 #pragma once
00006 #include "GameConfig.h"
00007 #include "SnakeList.h"
00008
00017     void Draw(const GameConfig& cfg, SnakeList& snake, int fruitX, int fruitY, int score);
00018 void Draw(const GameConfig& cfg, SnakeList& snake, int fruitX, int fruitY, int score);
```

4.5 GameConfig.cpp File Reference

Implementation of configuration loading.

```
#include "GameConfig.h"
#include <conio.h>
```

Functions

- [GameConfig loadConfig](#) (std::string filename)
Loads game configuration from a file.

4.5.1 Detailed Description

Implementation of configuration loading.

- Handles reading game parameters from external files and provides fallback default values in case of errors.

Definition in file [GameConfig.cpp](#).

4.5.2 Function Documentation

4.5.2.1 loadConfig()

```
GameConfig loadConfig (
    std::string filename)
```

Loads game configuration from a file.

Parameters

<i>filename</i>	Name of the file containing config data.
-----------------	--

Returns

[GameConfig](#) object with loaded or default values.

Definition at line 9 of file [GameConfig.cpp](#).

4.6 GameConfig.cpp

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

```
00001
00007 #include "GameConfig.h"
00008 #include <conio.h>
00009 GameConfig loadConfig(std::string filename){
00010     std::ifstream file(filename);
00011     GameConfig config;
00012     std::string tempDiff;
00013
00014     if (!file.is_open()) {
00015         std::cerr << "Error: cannot open the file: " << filename << "!" << std::endl;
00016         std::cerr << "Loading default settings..." << std::endl;
00017         std::cerr << "Press any key to continue..." << std::endl;
00018         _getch();
00019         system("cls");
00020         return { 20,15,100,Difficulty::EASY };
00021     }
00022
00023     file >> config.width >> config.height >> config.speed >> tempDiff;
00024     if (config.width > 100 || config.height > 24) {
00025         std::cerr << "Error: maximum width is 100 and maximum height is 24 " << std::endl;
00026         std::cerr << "Loading default settings..." << std::endl;
00027         std::cerr << "Press any key to continue..." << std::endl;
00028         _getch();
00029         system("cls");
00030
00031         return { 20,15,100,Difficulty::EASY };
00032     }
00033     if (tempDiff == "HARD") {
00034         config.diff = Difficulty::HARD;
00035     }
00036     else if (tempDiff == "MEDIUM") {
00037         config.diff = Difficulty::MEDIUM;
00038     }
00039     else {
00040         config.diff = Difficulty::EASY;
00041     }
00042     file.close();
00043     return config;
00044
00045
00046 }
```

4.7 GameConfig.h File Reference

Configuration structures and functions for game settings.

```
#include <iostream>
#include <windows.h>
#include <fstream>
#include "Types.h"
```

Classes

- struct [GameConfig](#)
Stores game parameters such as board dimensions and speed.

Functions

- [GameConfig loadConfig](#) (std::string filename)
Loads game configuration from a file.

4.7.1 Detailed Description

Configuration structures and functions for game settings.

Definition in file [GameConfig.h](#).

4.7.2 Function Documentation

4.7.2.1 loadConfig()

```
GameConfig loadConfig (  
    std::string filename)
```

Loads game configuration from a file.

Parameters

<i>filename</i>	Name of the file containing config data.
-----------------	--

Returns

[GameConfig](#) object with loaded or default values.

Definition at line 9 of file [GameConfig.cpp](#).

4.8 GameConfig.h

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

```
00001
00005 #pragma once
00006 #include <iostream>
00007 #include <windows.h>
00008 #include <fstream>
00009 #include "Types.h"
00010
00015 struct GameConfig {
00016     int width;
00017     int height;
00018     int speed;
00019     Difficulty diff;
00020 };
00021
00027 GameConfig loadConfig(std::string filename);
```

4.9 main.cpp File Reference

The main entry point for the Snake game.

```
#include "SnakeList.h"
#include "GameConfig.h"
#include "Draw.h"
#include "Types.h"
#include "SaveHighScore.h"
```

Functions

- int [main](#) ()

4.9.1 Detailed Description

The main entry point for the Snake game.

- Coordinates the game loop, input handling, movement logic, collision detection, and difficulty scaling.

Definition in file [main.cpp](#).

4.9.2 Function Documentation

4.9.2.1 main()

```
int main ()
```

Definition at line 14 of file [main.cpp](#).

4.10 main.cpp

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

```

00001
00007 #include "SnakeList.h"
00008 #include "GameConfig.h"
00009 #include "Draw.h"
00010 #include "Types.h"
00011 #include "SaveHighScore.h"
00012
00013
00014 int main() {
00015     GameConfig cfg = loadConfig("config.txt");
00016     eDirection dir = eDirection::STOP;
00017     SnakeList snake;
00018
00019     int score = 0;
00020     int headX = cfg.width / 2;
00021     int headY = cfg.height / 2;
00022     int fruitX = rand() % cfg.width;
00023     int fruitY = rand() % cfg.height;
00024     bool gameOver = false;
00025     snake.push_front(headX, headY);
00026
00027     while (!gameOver) {
00028
00029         if (_kbhit()) {
00030             switch (_getch())
00031             {
00032                 case 'a':
00033                     if(dir!=eDirection::RIGHT) dir = eDirection::LEFT;
00034                     break;
00035                 case 'd':
00036                     if(dir!=eDirection::LEFT) dir = eDirection::RIGHT;
00037                     break;
00038                 case 'w':
00039                     if(dir!=eDirection::DOWN) dir = eDirection::UP;
00040                     break;
00041                 case 's':
00042                     if (dir != eDirection::UP) dir = eDirection::DOWN;
00043                     break;
00044                 case 'x':
00045                     gameOver = true;
00046                     break;
00047             }
00048         }
00049
00050
00051         if (dir != eDirection::STOP) {
00052             if (dir == eDirection::LEFT) {
00053                 headX--;
00054             }
00055             else if (dir == eDirection::RIGHT) {
00056                 headX++;
00057             }
00058             else if (dir == eDirection::UP) {
00059                 headY--;
00060             }
00061             else if (dir == eDirection::DOWN) {
00062                 headY++;
00063             }
00064             if (snake.isSnakeAt(headX, headY)) {
00065                 gameOver = true;
00066             }
00067             snake.push_front(headX, headY);
00068
00069
00070             if (headX == fruitX && headY == fruitY) {
00071
00072
00073
00074                 switch (cfg.diff) {
00075                     case Difficulty::EASY:
00076                         score += 10;
00077                         fruitX = rand() % cfg.width;
00078                         fruitY = rand() % cfg.height;
00079                         break;
00080                     case Difficulty::MEDIUM:
00081                         score += 20;
00082                         cfg.speed -= 2;
00083                         fruitX = rand() % cfg.width;
00084                         fruitY = rand() % cfg.height;
00085                         break;
00086
00087                     case Difficulty::HARD:

```

```

00088         system("cls");
00089         score += 50;
00090         cfg.speed -= 3;
00091         if (cfg.width > 10) cfg.width -= 1;
00092         if (cfg.height > 5) cfg.height -= 1;
00093         fruitX = 1 + (rand() % (cfg.width - 2));
00094         fruitY = 1 + (rand() % (cfg.height - 2));
00095         break;
00096     }
00097     }
00098     MessageBeep (MB_ICONWARNING);
00099
00100
00101     }
00102     else {
00103         snake.pop_back();
00104     }
00105
00106 }
00107 Draw(cfg, snake, fruitX, fruitY, score);
00108 if (headX < 0 || headX >= cfg.width || headY < 0 || headY >= cfg.height) {
00109     gameOver = true;
00110
00111 }
00112 Sleep(cfg.speed);
00113
00114
00115
00116 }
00117 int finalScore = score * (110 - cfg.speed);
00118 std::cout << "===== " << std::endl;
00119 std::cout << "                GAME OVER!                " << std::endl;
00120 std::cout << "                Your Final Score: " << finalScore << std::endl;
00121 std::cout << "===== " << std::endl;
00122 std::string name;
00123 std::cout << "Enter your name: ";
00124 std::cin >> name;
00125 saveHighScore(name, finalScore);
00126 displayLeaderboard();
00127 std::cout << "\nPress any key to exit...";
00128 _getch();
00129 return 0;
00130 }

```

4.11 SaveHighScore.cpp File Reference

Implementation of the high score system.

```

#include "SaveHighScore.h"
#include <iostream>

```

Functions

- void [saveHighScore](#) (std::string playerName, int score)
Appends a new score to the highscore file.
- void [displayLeaderboard](#) ()
Displays the leaderboard from the highscore file to the console.

4.11.1 Detailed Description

Implementation of the high score system.

- Manages permanent storage of player results in a text file and handles leaderboard display.

Definition in file [SaveHighScore.cpp](#).

4.11.2 Function Documentation

4.11.2.1 displayLeaderboard()

```
void displayLeaderboard ()
```

Displays the leaderboard from the highscore file to the console.

Definition at line 22 of file [SaveHighScore.cpp](#).

4.11.2.2 saveHighScore()

```
void saveHighScore (
    std::string playerName,
    int score)
```

Appends a new score to the highscore file.

Parameters

<i>playerName</i>	Name of the player.
<i>score</i>	Final score achieved.

Definition at line 10 of file [SaveHighScore.cpp](#).

4.12 SaveHighScore.cpp

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

```
00001
00007 #pragma once
00008 #include "SaveHighScore.h"
00009 #include <iostream>
00010 void saveHighScore(std::string playerName, int score) {
00011     std::ofstream outFile("highscores.txt", std::ios::app);
00012     if (outFile.is_open()) {
00013         outFile << playerName << " | Score: " << score << std::endl;
00014         outFile.close();
00015     }
00016     else {
00017         std::cerr << "Error: Could not open highscores.txt for writing!";
00018     }
00019 }
00020
00021
00022 void displayLeaderboard(){
00023     std::ifstream inFile("highscores.txt");
00024     std::string line;
00025     system("cls");
00026     std::cout << "=== RECENT SCORES ===\n" << std::endl;
00027     if (inFile.is_open()) {
00028         while (getline(inFile, line)) {
00029             std::cout << line << std::endl;
00030         }
00031     }
00032     else {
00033         std::cout << "No scores yet. Be the first!" << std::endl;
00034     }
00035     std::cout << "\n===== " << std::endl;
00036
00037 }
```

4.13 SaveHighScore.h File Reference

File I/O operations for saving and displaying player scores.

```
#include <fstream>
#include <string>
```

Functions

- void [saveHighScore](#) (std::string playerName, int score)
Appends a new score to the highscore file.
- void [displayLeaderboard](#) ()
Displays the leaderboard from the highscore file to the console.

4.13.1 Detailed Description

File I/O operations for saving and displaying player scores.

Definition in file [SaveHighScore.h](#).

4.13.2 Function Documentation

4.13.2.1 displayLeaderboard()

```
void displayLeaderboard ()
```

Displays the leaderboard from the highscore file to the console.

Definition at line 22 of file [SaveHighScore.cpp](#).

4.13.2.2 saveHighScore()

```
void saveHighScore (
    std::string playerName,
    int score)
```

Appends a new score to the highscore file.

Parameters

<i>playerName</i>	Name of the player.
<i>score</i>	Final score achieved.

Definition at line 10 of file [SaveHighScore.cpp](#).

4.14 SaveHighScore.h

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

```
00001
00005 #pragma once
00006 #include <fstream>
00007 #include <string>
00013 void saveHighScore(std::string playerName, int score);
00014
00018 void displayLeaderboard();
```

4.15 SnakeList.cpp File Reference

Implementation of the [SnakeList](#) class and its memory management.

```
#include "SnakeList.h"
```

4.15.1 Detailed Description

Implementation of the [SnakeList](#) class and its memory management.

- Contains the logic for the doubly linked list, including the Rule of Five (copy/move constructors and assignment operators) and the sorting algorithm.

Definition in file [SnakeList.cpp](#).

4.16 SnakeList.cpp

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

```
00001
00007 #include "SnakeList.h"
00008
00009 Node::Node(int _x, int _y)
00010 : x(_x), y(_y), prev(nullptr), next(nullptr) {};
00011 SnakeList::SnakeList() : head(nullptr), tail(nullptr) {};
00012 void SnakeList::push_front(int x, int y) {
00013     auto newNode = std::make_unique<Node>(x, y);
00014     if (!head) {
00015         head = move(newNode);
00016         tail = head.get();
00017     }
00018     else {
00019         newNode->next = move(head);
00020         newNode->next->prev = newNode.get();
00021         head = move(newNode);
00022     }
00023
00024 }
00025
00026
00027 void SnakeList::pop_back() {
00028     if (!head) {
00029         return;
00030     }
00031     if (head.get() == tail) {
00032         head.reset();
00033         tail = nullptr;
00034     }
00035     else {
00036         Node* newTail = tail->prev;
```

```

00037         newTail->next.reset();
00038         tail = newTail;
00039     }
00040 }
00041 }
00042 bool SnakeList::isSnakeAt(int x, int y) const{
00043     Node* temp = head.get();
00044     while (temp) {
00045         if (temp->x == x && temp->y == y) {
00046             return true;
00047         }
00048         temp = temp->next.get();
00049     }
00050     return false;
00051 }
00052 }
00053 //Deep copy
00054 SnakeList::SnakeList(const SnakeList& other) : head(nullptr), tail(nullptr) {
00055     if (!other.tail) return;
00056     Node* temp = other.tail;
00057     while (temp) {
00058         this->push_front(temp->x, temp->y);
00059         temp = temp->prev;
00060     }
00061 }
00062 }
00063 //Move Constructor
00064 SnakeList::SnakeList(SnakeList&& other) noexcept : head(std::move(other.head)), tail(other.tail) {
00065     other.tail = nullptr;
00066 }
00067 }
00068 //Copy operator
00069 SnakeList& SnakeList::operator=(const SnakeList& other) {
00070     if (this != &other) {
00071         head.reset();
00072         tail = nullptr;
00073     }
00074     if (other.tail) {
00075         Node* temp = other.tail;
00076         while (temp) {
00077             this->push_front(temp->x, temp->y);
00078             temp = temp->prev;
00079         }
00080     }
00081 }
00082 }
00083 return *this;
00084 }
00085 }
00086 //Move operator
00087 SnakeList& SnakeList::operator=(SnakeList&& other) noexcept {
00088     if (this != &other) {
00089         head.reset();
00090         head = std::move(other.head);
00091         tail = other.tail;
00092         other.tail = nullptr;
00093     }
00094     return *this;
00095 }
00096 //sort method
00097 void SnakeList::sortContent() {
00098     if (!head || !head->next) return;
00099     bool swapped;
00100     do {
00101         swapped = false;
00102         Node* curr = head.get();
00103         while (curr->next) {
00104             if (curr->x > curr->next->x) {
00105                 std::swap(curr->x, curr->next->x);
00106                 std::swap(curr->y, curr->next->y);
00107                 swapped = true;
00108             }
00109             curr = curr->next.get();
00110         }
00111     } while (swapped);
00112 }
00113 }
00114 }

```

4.17 SnakeList.h File Reference

Header file containing the Doubly Linked List implementation for the Snake game.

```
#include <memory>
#include <iostream>
#include <conio.h>
#include <Windows.h>
```

Classes

- struct [Node](#)
Represents a single segment of the snake's body.
- class [SnakeList](#)
Doubly linked list for managing snake segments. Implements the Rule of Five and custom sorting/searching.

4.17.1 Detailed Description

Header file containing the Doubly Linked List implementation for the Snake game.

Definition in file [SnakeList.h](#).

4.18 SnakeList.h

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

```
00001
00005 #pragma once
00006 #include <memory>
00007 #include <iostream>
00008 #include <conio.h>
00009 #include <Windows.h>
00010
00015 struct Node {
00016     int x, y;
00017     std::unique_ptr<Node> next;
00018     Node* prev;
00019
00020     Node(int _x, int _y);
00021 };
00022
00028 class SnakeList {
00029     std::unique_ptr<Node> head;
00030     Node* tail;
00031
00032 public:
00033     SnakeList();
00034
00040     void push_front(int x, int y);
00041
00043     void pop_back();
00044
00049     bool isSnakeAt(int x, int y) const;
00050
00051     // Rule of Five implementations
00052     SnakeList(const SnakeList& other);
00053     SnakeList(SnakeList&& other) noexcept;
00054     SnakeList& operator=(const SnakeList& other);
00055     SnakeList& operator=(SnakeList&& other) noexcept;
00056
00058     void sortContent();
00059 };
00060
00061
00062
```

4.19 Types.h File Reference

Common enumerations used across the project.

Enumerations

- enum class `eDirection` {
 `STOP` = 0 , `LEFT` , `RIGHT` , `UP` ,
 `DOWN` }
Possible movement directions for the snake.
- enum class `Difficulty` { `EASY` , `MEDIUM` , `HARD` }
Difficulty levels that scale gameplay mechanics.

4.19.1 Detailed Description

Common enumerations used across the project.

Definition in file `Types.h`.

4.19.2 Enumeration Type Documentation

4.19.2.1 Difficulty

```
enum class Difficulty [strong]
```

`Difficulty` levels that scale gameplay mechanics.

Enumerator

EASY	
MEDIUM	
HARD	

Definition at line 15 of file `Types.h`.

4.19.2.2 eDirection

```
enum class eDirection [strong]
```

Possible movement directions for the snake.

Enumerator

STOP	
LEFT	

RIGHT	
UP	
DOWN	

Definition at line 7 of file [Types.h](#).

4.20 Types.h

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

```
00001
00005 #pragma once
00007 enum class eDirection {
00008     STOP = 0,
00009     LEFT,
00010     RIGHT,
00011     UP,
00012     DOWN
00013 };
00015 enum class Difficulty {
00016     EASY,
00017     MEDIUM,
00018     HARD
00019 };
```

Index

- diff
 - GameConfig, 5
- Difficulty
 - Types.h, 25
- displayLeaderboard
 - SaveHighScore.cpp, 20
 - SaveHighScore.h, 21
- DOWN
 - Types.h, 26
- Draw
 - Draw.cpp, 12
 - Draw.h, 13
- Draw.cpp, 11
 - Draw, 12
- Draw.h, 13
 - Draw, 13
- EASY
 - Types.h, 25
- eDirection
 - Types.h, 25
- GameConfig, 5
 - diff, 5
 - height, 5
 - speed, 6
 - width, 6
- GameConfig.cpp, 14
 - loadConfig, 15
- GameConfig.h, 16
 - loadConfig, 16
- HARD
 - Types.h, 25
- height
 - GameConfig, 5
- isSnakeAt
 - SnakeList, 9
- LEFT
 - Types.h, 25
- loadConfig
 - GameConfig.cpp, 15
 - GameConfig.h, 16
- main
 - main.cpp, 17
- main.cpp, 17
 - main, 17
- MEDIUM
 - Types.h, 25
- next
 - Node, 7
- Node, 6
 - next, 7
 - Node, 7
 - prev, 7
 - x, 7
 - y, 7
- operator=
 - SnakeList, 9
- pop_back
 - SnakeList, 9
- prev
 - Node, 7
- push_front
 - SnakeList, 9
- RIGHT
 - Types.h, 26
- saveHighScore
 - SaveHighScore.cpp, 20
 - SaveHighScore.h, 21
- SaveHighScore.cpp, 19
 - displayLeaderboard, 20
 - saveHighScore, 20
- SaveHighScore.h, 21
 - displayLeaderboard, 21
 - saveHighScore, 21
- SnakeList, 8
 - isSnakeAt, 9
 - operator=, 9
 - pop_back, 9
 - push_front, 9
 - SnakeList, 8
 - sortContent, 10
- SnakeList.cpp, 22
- SnakeList.h, 23
- sortContent
 - SnakeList, 10
- speed
 - GameConfig, 6
- STOP
 - Types.h, 25
- Types.h, 25
 - Difficulty, 25

DOWN, [26](#)
EASY, [25](#)
eDirection, [25](#)
HARD, [25](#)
LEFT, [25](#)
MEDIUM, [25](#)
RIGHT, [26](#)
STOP, [25](#)
UP, [26](#)

UP

Types.h, [26](#)

width

GameConfig, [6](#)

x

Node, [7](#)

y

Node, [7](#)