

# Noughts and Crosses

1.0

Generated by Doxygen 1.16.1



---

<b>1 Noughts and Crosses Documentation</b>	<b>1</b>
<b>2 File Index</b>	<b>3</b>
2.1 File List . . . . .	3
<b>3 File Documentation</b>	<b>5</b>
3.1 main.cpp File Reference . . . . .	5
3.1.1 Detailed Description . . . . .	6
3.1.2 Function Documentation . . . . .	7
3.1.2.1 checkDirection() . . . . .	7
3.1.2.2 checkWin() . . . . .	8
3.1.2.3 hardBotMove() . . . . .	8
3.1.2.4 initializeBoard() . . . . .	8
3.1.2.5 isDraw() . . . . .	9
3.1.2.6 isWinningCell() . . . . .	9
3.1.2.7 main() . . . . .	9
3.1.2.8 mediumBotMove() . . . . .	10
3.1.2.9 printBoard() . . . . .	10
3.1.2.10 printCell() . . . . .	10
3.1.2.11 randomMove() . . . . .	10
3.1.2.12 resetColor() . . . . .	11
3.1.2.13 rng() . . . . .	11
3.1.2.14 setColor() . . . . .	11
3.1.3 Variable Documentation . . . . .	11
3.1.3.1 board . . . . .	11
3.1.3.2 boardSize . . . . .	11
3.1.3.3 COLOR_BLUE . . . . .	12
3.1.3.4 COLOR_DEFAULT . . . . .	12
3.1.3.5 COLOR_GREEN . . . . .	12
3.1.3.6 COLOR_RED . . . . .	12
3.1.3.7 hConsole . . . . .	12
3.1.3.8 rd . . . . .	12
3.1.3.9 winLength . . . . .	13
3.1.3.10 winningCells . . . . .	13
3.2 main.cpp . . . . .	13



## Chapter 1

# Noughts and Crosses Documentation

This documentation describes the internal structure and logic of the Noughts and Crosses game implemented in C++.

The game supports dynamic board sizes, configurable win conditions, and multiple AI difficulty levels. It is intended as an academic demonstration of structured programming, algorithmic reasoning, and basic artificial intelligence techniques.



# Chapter 2

## File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	Noughts and Crosses (Tic-Tac-Toe) game with variable board size and AI bots . . . . .	5
--------------------------	---	---



# Chapter 3

## File Documentation

### 3.1 main.cpp File Reference

Noughts and Crosses (Tic-Tac-Toe) game with variable board size and AI bots.

```
#include <iostream>
#include <vector>
#include <random>
#include <utility>
#include <algorithm>
#include <string>
#include <ranges>
#include <windows.h>
```

#### Functions

- **mt19937 rng (rd())**  
*Mersenne Twister random engine.*
- **void setColor (WORD color)**  
*Sets the console text color.*
- **void resetColor ()**  
*Resets the console text color to default.*
- **void initializeBoard ()**  
*Initializes or resets the game board.*
- **bool isWinningCell (int r, int c)**  
*Checks if a given cell is part of the winning line.*
- **void printCell (char ch, bool winning)**  
*Prints a single cell with appropriate coloring.*
- **void printBoard ()**  
*Prints the entire game board to the console.*
- **bool checkDirection (int r, int c, int dr, int dc, char p)**  
*Checks for a winning sequence in a given direction.*
- **bool checkWin (char p)**  
*Checks if a player has won the game.*
- **bool isDraw ()**

- int `randomMove ()`  
*Generates a random valid move.*
- int `mediumBotMove (char bot, char human)`  
*Medium difficulty bot logic.*
- int `hardBotMove (char bot, char human)`  
*Hard difficulty bot logic.*
- int `main ()`  
*Program entry point.*

## Variables

- constexpr WORD `COLOR_DEFAULT` = FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE  
*Default console text color.*
- constexpr WORD `COLOR_RED` = FOREGROUND\_RED | FOREGROUND\_INTENSITY  
*Console color for player X.*
- constexpr WORD `COLOR_BLUE` = FOREGROUND\_BLUE | FOREGROUND\_INTENSITY  
*Console color for player O.*
- constexpr WORD `COLOR_GREEN` = FOREGROUND\_GREEN | FOREGROUND\_INTENSITY  
*Console color for winning cells.*
- HANDLE `hConsole` = GetStdHandle(STD\_OUTPUT\_HANDLE)  
*Handle to the Windows console.*
- int `boardSize`  
*Size of the game board ( $N \times N$ ).*
- int `winLength`  
*Number of consecutive symbols required to win.*
- vector<vector<char>> `board`  
*2D game board storing player symbols.*
- vector<pair<int, int>> `winningCells`  
*Stores coordinates of the winning cells.*
- random\_device `rd`  
*Random number generator device.*

### 3.1.1 Detailed Description

Noughts and Crosses (Tic-Tac-Toe) game with variable board size and AI bots.

This program implements a console-based noughts and crosses game supporting:

- Two-player mode
- Single-player mode vs AI bot
- Variable board sizes (3x3 up to 19x19)
- Dynamic win lengths
- Colored console output (Windows only)

The project demonstrates:

- Grid-based game logic
- Win detection algorithms
- Randomized and heuristic-based AI
- Console UI handling

**Author**

<Karthik Raman Keerangudi Kalayanaraman>

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

### 3.1.2 Function Documentation

#### 3.1.2.1 checkDirection()

```
bool checkDirection (
    int r,
    int c,
    int dr,
    int dc,
    char p)
```

Checks for a winning sequence in a given direction.

**Parameters**

<i>r</i>	Starting row.
<i>c</i>	Starting column.
<i>dr</i>	Row direction increment.
<i>dc</i>	Column direction increment.
<i>p</i>	Player symbol.

**Returns**

true if a winning sequence is found.

Definition at line [220](#) of file [main.cpp](#).

### 3.1.2.2 checkWin()

```
bool checkWin (
    char p)
```

Checks if a player has won the game.

#### Parameters

<i>p</i>	Player symbol.
----------	----------------

#### Returns

true if the player has won.

Definition at line [244](#) of file [main.cpp](#).

### 3.1.2.3 hardBotMove()

```
int hardBotMove (
    char bot,
    char human)
```

Hard difficulty bot logic.

#### Parameters

<i>bot</i>	Bot symbol.
<i>human</i>	Human player symbol.

#### Returns

Selected move.

Definition at line [332](#) of file [main.cpp](#).

### 3.1.2.4 initializeBoard()

```
void initializeBoard ()
```

Initializes or resets the game board.

Definition at line [133](#) of file [main.cpp](#).

### 3.1.2.5 isDraw()

```
bool isDraw ()
```

Checks if the game has ended in a draw.

#### Returns

true if the board is full and no winner exists.

Definition at line [262](#) of file [main.cpp](#).

### 3.1.2.6 isWinningCell()

```
bool isWinningCell (
    int r,
    int c)
```

Checks if a given cell is part of the winning line.

#### Parameters

<i>r</i>	Row index.
<i>c</i>	Column index.

#### Returns

true if the cell belongs to the winning combination.

Definition at line [148](#) of file [main.cpp](#).

### 3.1.2.7 main()

```
int main ()
```

Program entry point.

#### Returns

Exit status code.

Definition at line [349](#) of file [main.cpp](#).

### 3.1.2.8 mediumBotMove()

```
int mediumBotMove (
    char bot,
    char human)
```

Medium difficulty bot logic.

#### Parameters

<i>bot</i>	Bot symbol.
<i>human</i>	Human player symbol.

#### Returns

Selected move.

Definition at line 300 of file [main.cpp](#).

### 3.1.2.9 printBoard()

```
void printBoard ()
```

Prints the entire game board to the console.

Definition at line 174 of file [main.cpp](#).

### 3.1.2.10 printCell()

```
void printCell (
    char ch,
    bool winning)
```

Prints a single cell with appropriate coloring.

#### Parameters

<i>ch</i>	Character to print.
<i>winning</i>	Whether the cell is part of a winning line.

Definition at line 162 of file [main.cpp](#).

### 3.1.2.11 randomMove()

```
int randomMove ()
```

Generates a random valid move.

#### Returns

Cell number of the chosen move.

Definition at line 277 of file [main.cpp](#).

### 3.1.2.12 resetColor()

```
void resetColor ()
```

Resets the console text color to default.

Definition at line 122 of file [main.cpp](#).

### 3.1.2.13 rng()

```
mt19937 rng (
    rd() )
```

Mersenne Twister random engine.

### 3.1.2.14 setColor()

```
void setColor (
    WORD color)
```

Sets the console text color.

#### Parameters

<i>color</i>	Windows color attribute.
--------------	--------------------------

Definition at line 115 of file [main.cpp](#).

## 3.1.3 Variable Documentation

### 3.1.3.1 board

```
vector<vector<char>> board
```

2D game board storing player symbols.

Definition at line 90 of file [main.cpp](#).

### 3.1.3.2 boardSize

```
int boardSize
```

Size of the game board (N x N).

Definition at line 80 of file [main.cpp](#).

### 3.1.3.3 COLOR\_BLUE

```
WORD COLOR_BLUE = FOREGROUND_BLUE | FOREGROUND_INTENSITY [constexpr]
```

Console color for player O.

Definition at line [61](#) of file [main.cpp](#).

### 3.1.3.4 COLOR\_DEFAULT

```
WORD COLOR_DEFAULT = FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE [constexpr]
```

Default console text color.

Definition at line [51](#) of file [main.cpp](#).

### 3.1.3.5 COLOR\_GREEN

```
WORD COLOR_GREEN = FOREGROUND_GREEN | FOREGROUND_INTENSITY [constexpr]
```

Console color for winning cells.

Definition at line [66](#) of file [main.cpp](#).

### 3.1.3.6 COLOR\_RED

```
WORD COLOR_RED = FOREGROUND_RED | FOREGROUND_INTENSITY [constexpr]
```

Console color for player X.

Definition at line [56](#) of file [main.cpp](#).

### 3.1.3.7 hConsole

```
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE)
```

Handle to the Windows console.

Definition at line [75](#) of file [main.cpp](#).

### 3.1.3.8 rd

```
random_device rd
```

Random number generator device.

Definition at line [100](#) of file [main.cpp](#).

### 3.1.3.9 winLength

```
int winLength
```

Number of consecutive symbols required to win.

Definition at line 85 of file [main.cpp](#).

### 3.1.3.10 winningCells

```
vector<pair<int, int> > winningCells
```

Stores coordinates of the winning cells.

Definition at line 95 of file [main.cpp](#).

## 3.2 main.cpp

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

```
00001
00020
00032
00033 #include <iostream>
00034 #include <vector>
00035 #include <random>
00036 #include <utility>
00037 #include <algorithm>
00038 #include <string>
00039 #include <ranges>
00040 #include <windows.h>
00041
00042 using namespace std;
00043
00044 /* ----- */                                     */
00045 /*          Global Constants                   */ */
00046 /* ----- */                                     */
00047
00051 constexpr WORD COLOR_DEFAULT = FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE;
00052
00056 constexpr WORD COLOR_RED = FOREGROUND_RED | FOREGROUND_INTENSITY;
00057
00061 constexpr WORD COLOR_BLUE = FOREGROUND_BLUE | FOREGROUND_INTENSITY;
00062
00066 constexpr WORD COLOR_GREEN = FOREGROUND_GREEN | FOREGROUND_INTENSITY;
00067
00068 /* ----- */                                     */
00069 /*          Global Variables                  */ */
00070 /* ----- */                                     */
00071
00075 HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
00076
00080 int boardSize;
00081
00085 int winLength;
00086
00090 vector<vector<char>> board;
00091
00095 vector<pair<int, int>> winningCells;
00096
00100 random_device rd;
00101
00105 mt19937 rng(rd());
00106
00107 /* ----- */                                     */
00108 /*          Console Utility Functions        */ */
00109 /* ----- */                                     */
00110
00115 void setColor(WORD color) {
00116     SetConsoleTextAttribute(hConsole, color);
00117 }
```

```

00118
00119 void resetColor() {
00120     setColor(COLOR_DEFAULT);
00121 }
00122
00123 /* ----- */                                     */
00124 /*          Game Initialization Logic           */
00125 /* ----- */                                     */
00126
00127
00128 /* ----- */                                     */
00129 /*          Rendering Functions                 */
00130 /* ----- */                                     */
00131
00132
00133 void initializeBoard() {
00134     board.assign(boardSize, vector<char>(boardSize, '.'));
00135     winningCells.clear();
00136 }
00137
00138 /* ----- */                                     */
00139 /*          Rendering Functions                 */
00140 /* ----- */                                     */
00141
00142 bool isWinningCell(int r, int c) {
00143     return std::ranges::any_of(
00144         winningCells,
00145         [&](const pair<int, int>& p) {
00146             return p.first == r && p.second == c;
00147         }
00148     );
00149 }
00150
00151
00152 void printCell(char ch, bool winning) {
00153     if (winning) setColor(COLOR_GREEN);
00154     else if (ch == 'X') setColor(COLOR_RED);
00155     else if (ch == 'O') setColor(COLOR_BLUE);
00156
00157     cout << ch;
00158     resetColor();
00159 }
00160
00161
00162 void printBoard() {
00163     cout << "\n";
00164
00165     int cell = 1;
00166
00167     for (int i = 0; i < boardSize; ++i) {
00168         constexpr size_t CELL_W = 5;
00169
00170         for (int j = 0; j < boardSize; ++j) {
00171
00172             if (board[i][j] == '.') {
00173                 const string s = to_string(cell);
00174                 const size_t len = s.size();
00175
00176                 const size_t left = (CELL_W - len) / 2;
00177                 const size_t right = CELL_W - len - left;
00178
00179                 cout << string(left, ' ') << s << string(right, ' ');
00180             } else {
00181                 constexpr size_t left = (CELL_W - 1) / 2;
00182                 constexpr size_t right = CELL_W - 1 - left;
00183
00184                 cout << string(left, ' ');
00185                 printCell(board[i][j], isWinningCell(i, j));
00186                 cout << string(right, ' ');
00187             }
00188
00189         }
00190
00191         ++cell;
00192     }
00193     cout << "\n";
00194 }
00195
00196
00197 /* ----- */                                     */
00198 /*          Win / Draw Detection Logic        */
00199 /* ----- */                                     */
00200
00201
00202 bool checkDirection(int r, int c, int dr, int dc, char p) {
00203     vector<pair<int, int>> temp;
00204
00205     for (int i = 0; i < winLength; ++i) {
00206         int nr = r + i * dr;
00207         int nc = c + i * dc;
00208
00209         if (nr < 0 || nr >= boardSize || nc < 0 || nc >= boardSize)
00210             return false;
00211         if (board[nr][nc] != p)
00212             return false;
00213
00214         temp.emplace_back(nr, nc);
00215     }
00216
00217     return temp.size() == winLength;
00218 }
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233

```

```

00234     winningCells = temp;
00235     return true;
00236 }
00237 }
00238
00239 bool checkWin(char p) {
00240     winningCells.clear();
00241
00242     for (int i = 0; i < boardSize; ++i)
00243         for (int j = 0; j < boardSize; ++j)
00244             if (board[i][j] == p) {
00245                 if (checkDirection(i, j, 0, 1, p)) return true;
00246                 if (checkDirection(i, j, 1, 0, p)) return true;
00247                 if (checkDirection(i, j, 1, 1, p)) return true;
00248                 if (checkDirection(i, j, 1, -1, p)) return true;
00249             }
00250     return false;
00251 }
00252
00253
00254 */
00255 /* ----- */ * AI Move Logic */ */
00256 /* ----- */
00257
00258 int randomMove() {
00259     vector<int> freeCells;
00260     int total = boardSize * boardSize;
00261
00262     for (int i = 0; i < total; ++i) {
00263         int r = i / boardSize;
00264         int c = i % boardSize;
00265         if (board[r][c] == '.')
00266             freeCells.push_back(i + 1);
00267     }
00268
00269     if (freeCells.empty()) return -1;
00270
00271     uniform_int_distribution<size_t> dist(0, freeCells.size() - 1);
00272     return freeCells[dist(rng)];
00273 }
00274
00275 int mediumBotMove(char bot, char human) {
00276     int total = boardSize * boardSize;
00277
00278     for (int i = 1; i <= total; ++i) {
00279         int r = (i - 1) / boardSize;
00280         int c = (i - 1) % boardSize;
00281         if (board[r][c] == '.') {
00282             board[r][c] = bot;
00283             if (checkWin(bot)) { board[r][c] = '.'; return i; }
00284             board[r][c] = '.';
00285         }
00286     }
00287
00288     for (int i = 1; i <= total; ++i) {
00289         int r = (i - 1) / boardSize;
00290         int c = (i - 1) % boardSize;
00291         if (board[r][c] == '.') {
00292             board[r][c] = human;
00293             if (checkWin(human)) { board[r][c] = '.'; return i; }
00294             board[r][c] = '.';
00295         }
00296     }
00297
00298     return randomMove();
00299 }
00300
00301 int hardBotMove(char bot, char human) {
00302     int center = (boardSize * boardSize) / 2 + 1;
00303     int r = (center - 1) / boardSize;
00304     int c = (center - 1) % boardSize;
00305
00306     if (board[r][c] == '.') return center;
00307     return mediumBotMove(bot, human);
00308 }
00309
00310 */
00311 /* ----- */ * Main */ */
00312 /* ----- */
00313
00314 int main() {

```

```

00350     while (true) {
00351         cout << "\n1 - Two Players\n"
00352             "2 - Single Player vs Bot\n"
00353             "404 - Exit\n"
00354             "Choice: ";
00355
00356         int mode;
00357         cin >> mode;
00358         if (mode == 404) break;
00359
00360         bool vsBot = (mode == 2);
00361         int difficulty = 0;
00362
00363         if (vsBot) {
00364             cout << "Bot Difficulty (1=Easy, 2=Medium, 3=Hard): ";
00365             cin >> difficulty;
00366         }
00367
00368         cout << "Enter board size (3 to 19): ";
00369         cin >> boardSize;
00370         if (boardSize < 3 || boardSize > 19) continue;
00371
00372         if (boardSize <= 6) winLength = 3;
00373         else if (boardSize <= 9) winLength = 4;
00374         else winLength = 5;
00375
00376         initializeBoard();
00377         char current = 'X';
00378
00379         while (true) {
00380             printBoard();
00381             cout << "\n0 - Restart | 404 - Exit Program\n";
00382
00383             int move;
00384             if (vsBot && current == 'O') {
00385                 if (difficulty == 1) move = randomMove();
00386                 else if (difficulty == 2) move = mediumBotMove('O', 'X');
00387                 else move = hardBotMove('O', 'X');
00388                 cout << "Bot chooses: " << move << "\n";
00389             } else {
00390                 cout << "Player " << current << " move: ";
00391                 cin >> move;
00392             }
00393
00394             if (move == 404) return 0;
00395             if (move == 0) break;
00396
00397             int total = boardSize * boardSize;
00398             if (move < 1 || move > total) continue;
00399
00400             int r = (move - 1) / boardSize;
00401             int c = (move - 1) % boardSize;
00402             if (board[r][c] != '.') continue;
00403
00404             board[r][c] = current;
00405
00406             if (checkWin(current)) {
00407                 printBoard();
00408                 cout << "\nPlayer " << current << " wins!\n";
00409                 break;
00410             }
00411
00412             if (isDraw()) {
00413                 printBoard();
00414                 cout << "\nDraw!\n";
00415                 break;
00416             }
00417
00418             current = (current == 'X') ? 'O' : 'X';
00419         }
00420     }
00421
00422     cout << "Program ended.\n";
00423     return 0;
00424 }
```