Noughts and Crosses

1.0

Generated by Doxygen 1.16.1

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:

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 ()

Checks if the game has ended in a draw.
- 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 x 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

r	Starting row.
c	Starting column.
dr	Row direction increment.
dc	Column direction increment.
p	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

p	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

bot	Bot symbol.
human	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

r	Row index.
c	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

bot	Bot symbol.
human	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

ch	Character to print.
winning	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

color	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 }
```

```

void resetColor() {
    setColor(COLOR_DEFAULT);
}

/* -------------------------------------------------------------------------- */
/*                          Game Initialization Logic                          */
/* -------------------------------------------------------------------------- */

void initializeBoard() {
    board.assign(boardSize, vector<char>(boardSize, '.'));
    winningCells.clear();
}

/* -------------------------------------------------------------------------- */
/*                             Rendering Functions                             */
/* -------------------------------------------------------------------------- */

bool isWinningCell(int r, int c) {
    return std::ranges::any_of(
        winningCells,
        [&](const pair<int, int>& p) {
            return p.first == r && p.second == c;
        }
    );
}

void printCell(char ch, bool winning) {
    if (winning) setColor(COLOR_GREEN);
    else if (ch == 'X') setColor(COLOR_RED);
    else if (ch == 'O') setColor(COLOR_BLUE);

    cout « ch;
    resetColor();
}

void printBoard() {
    cout « "\n";

    int cell = 1;

    for (int i = 0; i < boardSize; ++i) {
        constexpr size_t CELL_W = 5;

        for (int j = 0; j < boardSize; ++j) {

            if (board[i][j] == '.') {
                const string s = to_string(cell);
                const size_t len = s.size();

                const size_t left  = (CELL_W - len) / 2;
                const size_t right = CELL_W - len - left;

                cout « string(left, ' ') « s « string(right, ' ');
            } else {
                constexpr size_t left  = (CELL_W - 1) / 2;
                constexpr size_t right = CELL_W - 1 - left;

                cout « string(left, ' ');
                printCell(board[i][j], isWinningCell(i, j));
                cout « string(right, ' ');
            }

            ++cell;
        }
        cout « "\n";
    }
}

/* -------------------------------------------------------------------------- */
/*                          Win / Draw Detection Logic                         */
/* -------------------------------------------------------------------------- */

bool checkDirection(int r, int c, int dr, int dc, char p) {
    vector<pair<int, int» temp;

    for (int i = 0; i < winLength; ++i) {
        int nr = r + i * dr;
        int nc = c + i * dc;

        if (nr < 0 || nr >= boardSize || nc < 0 || nc >= boardSize)
            return false;
        if (board[nr][nc] != p)
            return false;

        temp.emplace_back(nr, nc);
    }
```

```cpp

    winningCells = temp;
    return true;
}

bool checkWin(char p) {
    winningCells.clear();

    for (int i = 0; i < boardSize; ++i)
        for (int j = 0; j < boardSize; ++j)
            if (board[i][j] == p) {
                if (checkDirection(i, j, 0, 1, p)) return true;
                if (checkDirection(i, j, 1, 0, p)) return true;
                if (checkDirection(i, j, 1, 1, p)) return true;
                if (checkDirection(i, j, 1, -1, p)) return true;
            }
    return false;
}

bool isDraw() {
    for (const auto& row : board)
        for (char c : row)
            if (c == '.') return false;
    return true;
}

/* ------------------------------------------------------------------------- */
/*                              AI Move Logic                                  */
/* ------------------------------------------------------------------------- */

int randomMove() {
    vector<int> freeCells;
    int total = boardSize * boardSize;

    for (int i = 0; i < total; ++i) {
        int r = i / boardSize;
        int c = i % boardSize;
        if (board[r][c] == '.')
            freeCells.push_back(i + 1);
    }

    if (freeCells.empty()) return -1;

    uniform_int_distribution<size_t> dist(0, freeCells.size() - 1);
    return freeCells[dist(rng)];
}

int mediumBotMove(char bot, char human) {
    int total = boardSize * boardSize;

    for (int i = 1; i <= total; ++i) {
        int r = (i - 1) / boardSize;
        int c = (i - 1) % boardSize;
        if (board[r][c] == '.') {
            board[r][c] = bot;
            if (checkWin(bot)) { board[r][c] = '.'; return i; }
            board[r][c] = '.';
        }
    }

    for (int i = 1; i <= total; ++i) {
        int r = (i - 1) / boardSize;
        int c = (i - 1) % boardSize;
        if (board[r][c] == '.') {
            board[r][c] = human;
            if (checkWin(human)) { board[r][c] = '.'; return i; }
            board[r][c] = '.';
        }
    }

    return randomMove();
}

int hardBotMove(char bot, char human) {
    int center = (boardSize * boardSize) / 2 + 1;
    int r = (center - 1) / boardSize;
    int c = (center - 1) % boardSize;

    if (board[r][c] == '.') return center;
    return mediumBotMove(bot, human);
}

/* ------------------------------------------------------------------------- */
/*                                   Main                                    */
/* ------------------------------------------------------------------------- */

int main() {
    while (true) {
        cout « "\n1 - Two Players\n"
                "2 - Single Player vs Bot\n"
                "404 - Exit\n"
                "Choice: ";

        int mode;
        cin » mode;
        if (mode == 404) break;

        bool vsBot = (mode == 2);
        int difficulty = 0;

        if (vsBot) {
            cout « "Bot Difficulty (1=Easy, 2=Medium, 3=Hard): ";
            cin » difficulty;
        }

        cout « "Enter board size (3 to 19): ";
        cin » boardSize;
        if (boardSize < 3 || boardSize > 19) continue;

        if (boardSize <= 6) winLength = 3;
        else if (boardSize <= 9) winLength = 4;
        else winLength = 5;

        initializeBoard();
        char current = 'X';

        while (true) {
            printBoard();
            cout « "\n0 - Restart | 404 - Exit Program\n";

            int move;
            if (vsBot && current == 'O') {
                if (difficulty == 1) move = randomMove();
                else if (difficulty == 2) move = mediumBotMove('O', 'X');
                else move = hardBotMove('O', 'X');
                cout « "Bot chooses: " « move « "\n";
            } else {
                cout « "Player " « current « " move: ";
                cin » move;
            }

            if (move == 404) return 0;
            if (move == 0) break;

            int total = boardSize * boardSize;
            if (move < 1 || move > total) continue;

            int r = (move - 1) / boardSize;
            int c = (move - 1) % boardSize;
            if (board[r][c] != '.') continue;

            board[r][c] = current;

            if (checkWin(current)) {
                printBoard();
                cout « "\nPlayer " « current « " wins!\n";
                break;
            }

            if (isDraw()) {
                printBoard();
                cout « "\nDraw!\n";
                break;
            }

            current = (current == 'X') ? 'O' : 'X';
        }
    }

    cout « "Program ended.\n";
    return 0;
}
```