REPORT

TICTACTOE USING MINIMAX IN C

About TicTacToe

Tic-Tac-Toe game involves 2 players placing their respective symbols in a 3x3 grid. The player who manages to place three of their symbols in horizontal/vertical/diagonal row wins the game. If either player fails to do so the game ends in a draw. If both the people always play their optimal strategies the game always ends in a draw.

Tic-Tac-Toe as a State Space

State spaces are good representations for board games such as Tic-Tac-Toe. The state of a game can be described by the contents of the board and the player whose turn is next. The board can be represented as an array of 9 cells, each of which may contain an X or O or be empty.

State:
Player to move next: X or O.

Board configuration:

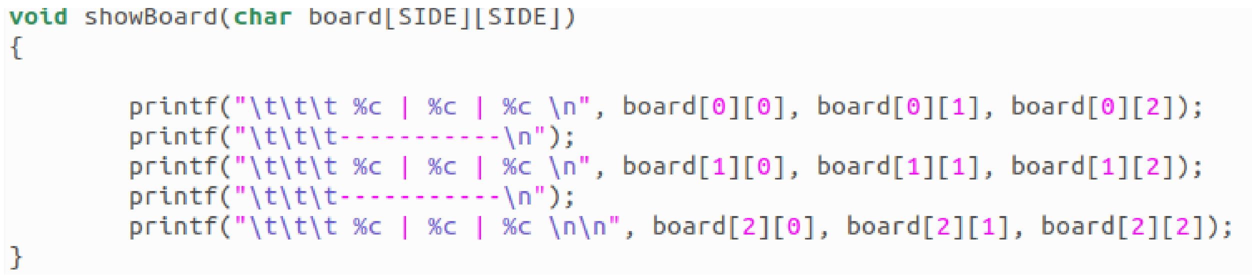
```
X     O
   O
X     X
```

Operators: Change an empty cell to X or O.

Start State: Board empty; X's turn.

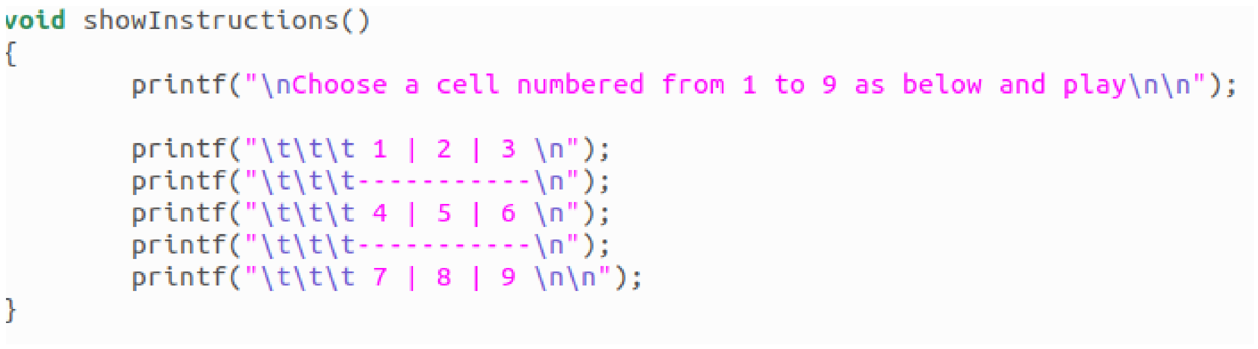
Terminal States:
Three X's in a row; Three O's in a row; All cells full.

A function to show the current board status



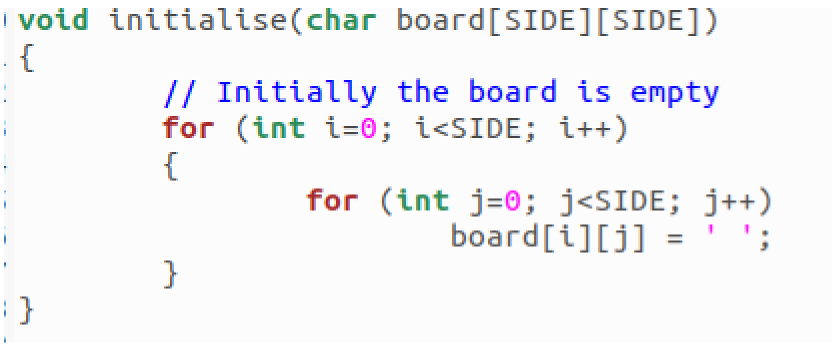
```c
void showBoard(char board[SIDE][SIDE])
{

        printf("\t\t\t %c | %c | %c \n", board[0][0], board[0][1], board[0][2]);
        printf("\t\t\t-----------\n");
        printf("\t\t\t %c | %c | %c \n", board[1][0], board[1][1], board[1][2]);
        printf("\t\t\t-----------\n");
        printf("\t\t\t %c | %c | %c \n\n", board[2][0], board[2][1], board[2][2]);
}
```

A function to show the instructions



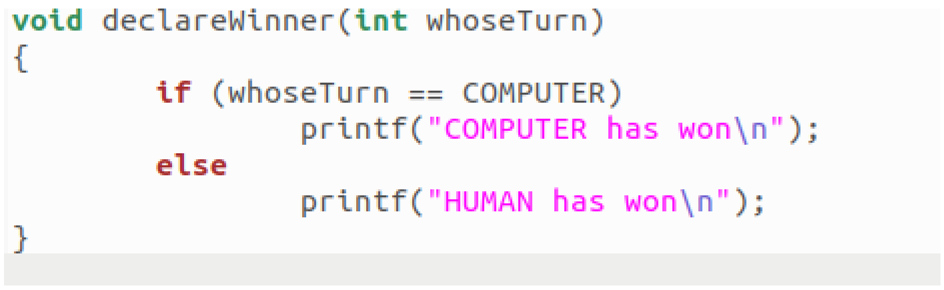
```c
void showInstructions()
{
        printf("\nChoose a cell numbered from 1 to 9 as below and play\n\n");

        printf("\t\t\t 1 | 2 | 3 \n");
        printf("\t\t\t-----------\n");
        printf("\t\t\t 4 | 5 | 6 \n");
        printf("\t\t\t-----------\n");
        printf("\t\t\t 7 | 8 | 9 \n\n");
}
```

A function to initialize the game



```c
void initialise(char board[SIDE][SIDE])
{
        // Initially the board is empty
        for (int i=0; i<SIDE; i++)
        {
                for (int j=0; j<SIDE; j++)
                        board[i][j] = ' ';
        }
}
```

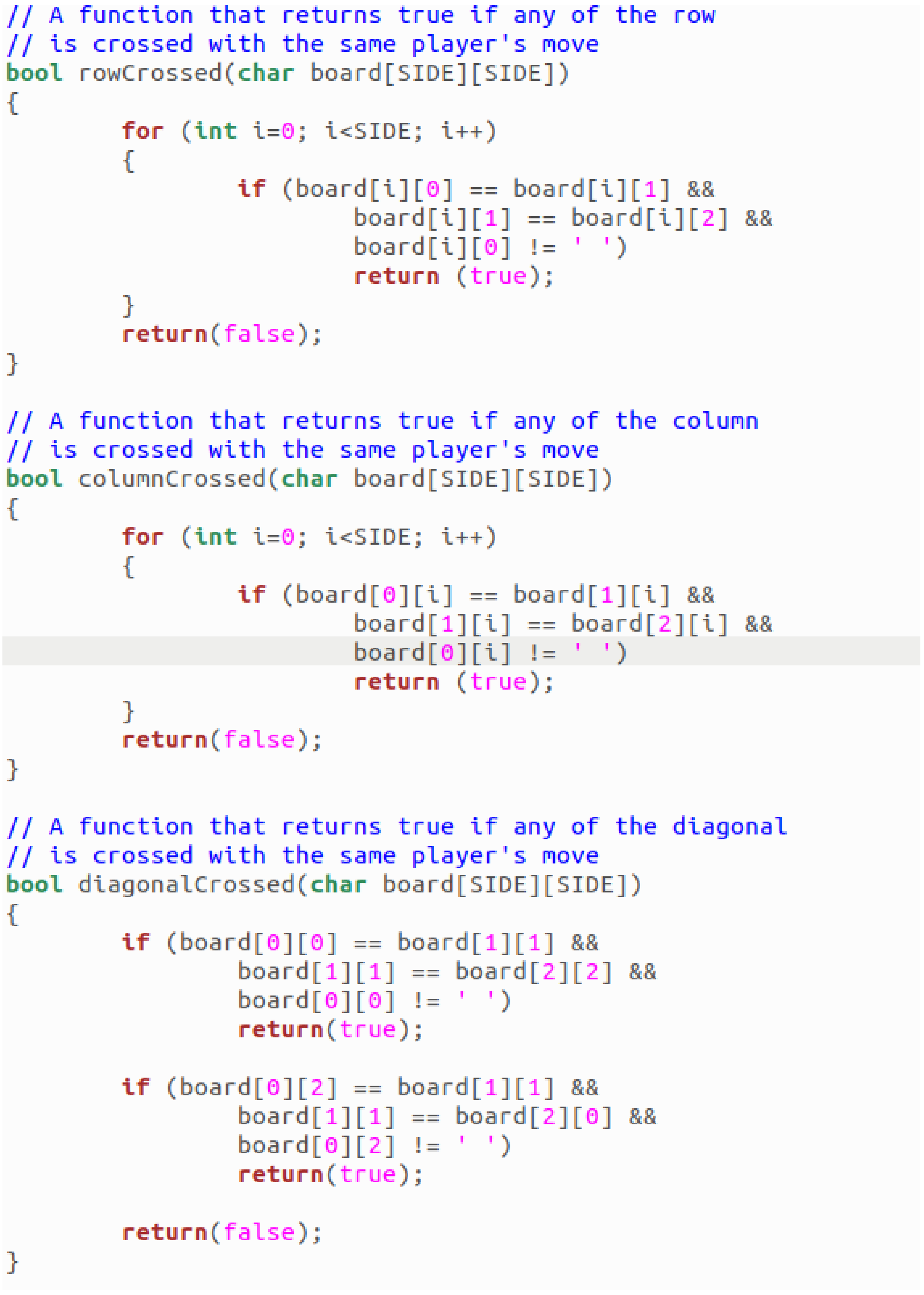
A function to declare the winner of the game



```
void declareWinner(int whoseTurn)
{
        if (whoseTurn == COMPUTER)
                printf("COMPUTER has won\n");
        else
                printf("HUMAN has won\n");
}
```

A function that returns true if any of the row or column or diagonally is crossed with the same player's move respectively.

The player who manages to place three of their symbols in horizontal/vertical/diagonal row wins the game.rowCrossed(), columnCrossed() and diagonalCrossed() functions checks for horizontal,vertical and diagonal wins respectively.



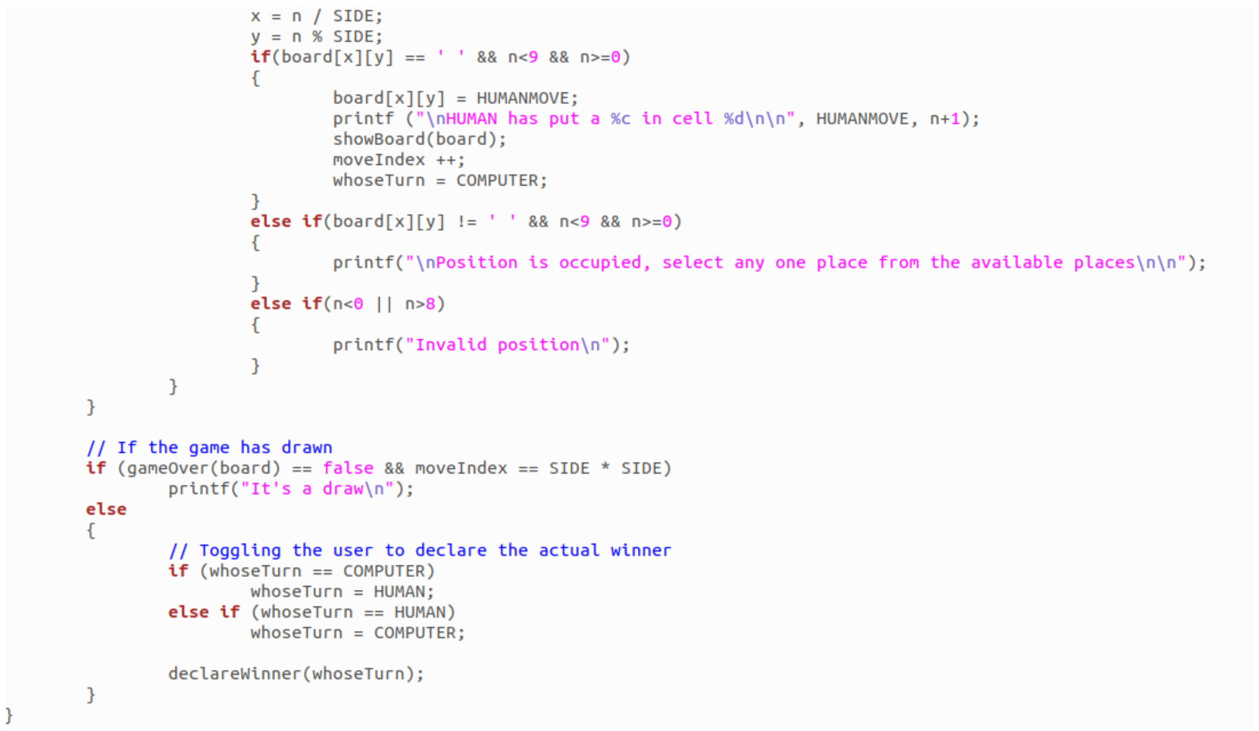
```
// A function that returns true if any of the row
// is crossed with the same player's move
bool rowCrossed(char board[SIDE][SIDE])
{
        for (int i=0; i<SIDE; i++)
        {
                if (board[i][0] == board[i][1] &&
                        board[i][1] == board[i][2] &&
                        board[i][0] != ' ')
                        return (true);
        }
        return(false);
}

// A function that returns true if any of the column
// is crossed with the same player's move
bool columnCrossed(char board[SIDE][SIDE])
{
        for (int i=0; i<SIDE; i++)
        {
                if (board[0][i] == board[1][i] &&
                        board[1][i] == board[2][i] &&
                        board[0][i] != ' ')
                        return (true);
        }
        return(false);
}

// A function that returns true if any of the diagonal
// is crossed with the same player's move
bool diagonalCrossed(char board[SIDE][SIDE])
{
        if (board[0][0] == board[1][1] &&
                board[1][1] == board[2][2] &&
                board[0][0] != ' ')
                return(true);

        if (board[0][2] == board[1][1] &&
                board[1][1] == board[2][0] &&
                board[0][2] != ' ')
                return(true);

        return(false);
}
```

A function that returns true if the game is over else it returns a false

```
bool gameOver(char board[SIDE][SIDE])
{
        return(rowCrossed(board) || columnCrossed(board) || diagonalCrossed(board) );
}
```

playticTacToe takes an argument whoseTurn,which is a user input choice. If the choice is 'n' , turn is COMPUTER else HUMAN. This function toggles between COMPUTER and HUMAN everytime as long as the game is not drawn.When it's COMPUTER turn, the function calls **bestMove()** function to obtain the best position.When it's HUMAN turn,he is asked to enter the position.

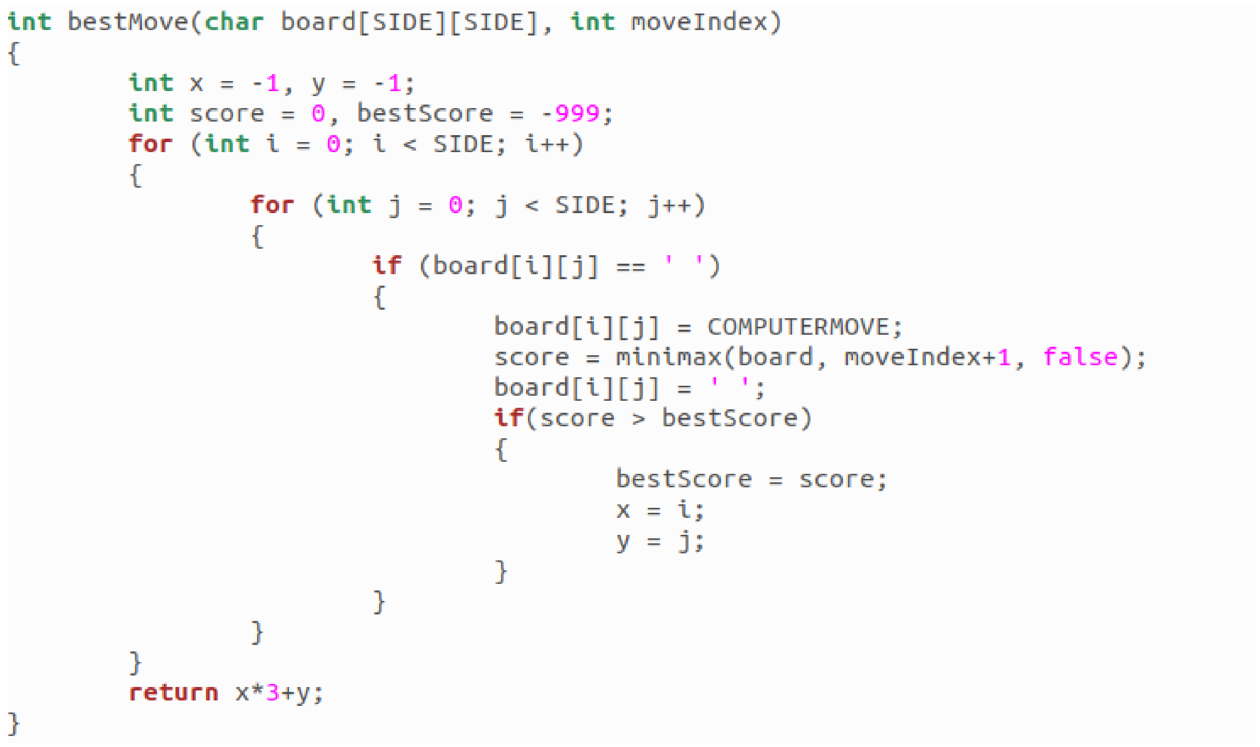
```
void playTicTacToe(int whoseTurn)
{
        char board[SIDE][SIDE];
        int moveIndex = 0, x = 0, y = 0;

        initialise(board);
        showInstructions();

        // Keep playing till the game is over or it is a draw
        while (gameOver(board) == false && moveIndex != SIDE*SIDE)
        {
                int n;
                if (whoseTurn == COMPUTER)
                {
                        n = bestMove(board, moveIndex);
                        x = n / SIDE;
                        y = n % SIDE;
                        board[x][y] = COMPUTERMOVE;
                        printf("COMPUTER has put a %c in cell %d\n\n", COMPUTERMOVE, n+1);
                        showBoard(board);
                        moveIndex ++;
                        whoseTurn = HUMAN;
                }

                else if (whoseTurn == HUMAN)
                {
                        printf("You can insert in the following positions : ");
                        for(int i=0; i<SIDE; i++)
                                for (int j = 0; j < SIDE; j++)
                                        if (board[i][j] == ' ')
                                                printf("%d ", (i * 3 + j) + 1);
                        printf("\n\nEnter the position = ");
                        scanf("%d",&n);
                        n--;
```



```
                        x = n / SIDE;
                        y = n % SIDE;
                        if(board[x][y] == ' ' && n<9 && n>=0)
                        {
                                board[x][y] = HUMANMOVE;
                                printf ("\nHUMAN has put a %c in cell %d\n\n", HUMANMOVE, n+1);
                                showBoard(board);
                                moveIndex ++;
                                whoseTurn = COMPUTER;
                        }
                        else if(board[x][y] != ' ' && n<9 && n>=0)
                        {
                                printf("\nPosition is occupied, select any one place from the available places\n\n");
                        }
                        else if(n<0 || n>8)
                        {
                                printf("Invalid position\n");
                        }
                }
        }

        // If the game has drawn
        if (gameOver(board) == false && moveIndex == SIDE * SIDE)
                printf("It's a draw\n");
        else
        {
                // Toggling the user to declare the actual winner
                if (whoseTurn == COMPUTER)
                        whoseTurn = HUMAN;
                else if (whoseTurn == HUMAN)
                        whoseTurn = COMPUTER;

                declareWinner(whoseTurn);
        }
}
```

Function to calculate best move

The bestMove function calls minimax for every empty position on board to calculate score,which is then compared with bestScore which is initialized to -999.If score is greater than bestScore , bestScore is assigned to score and returns the position.

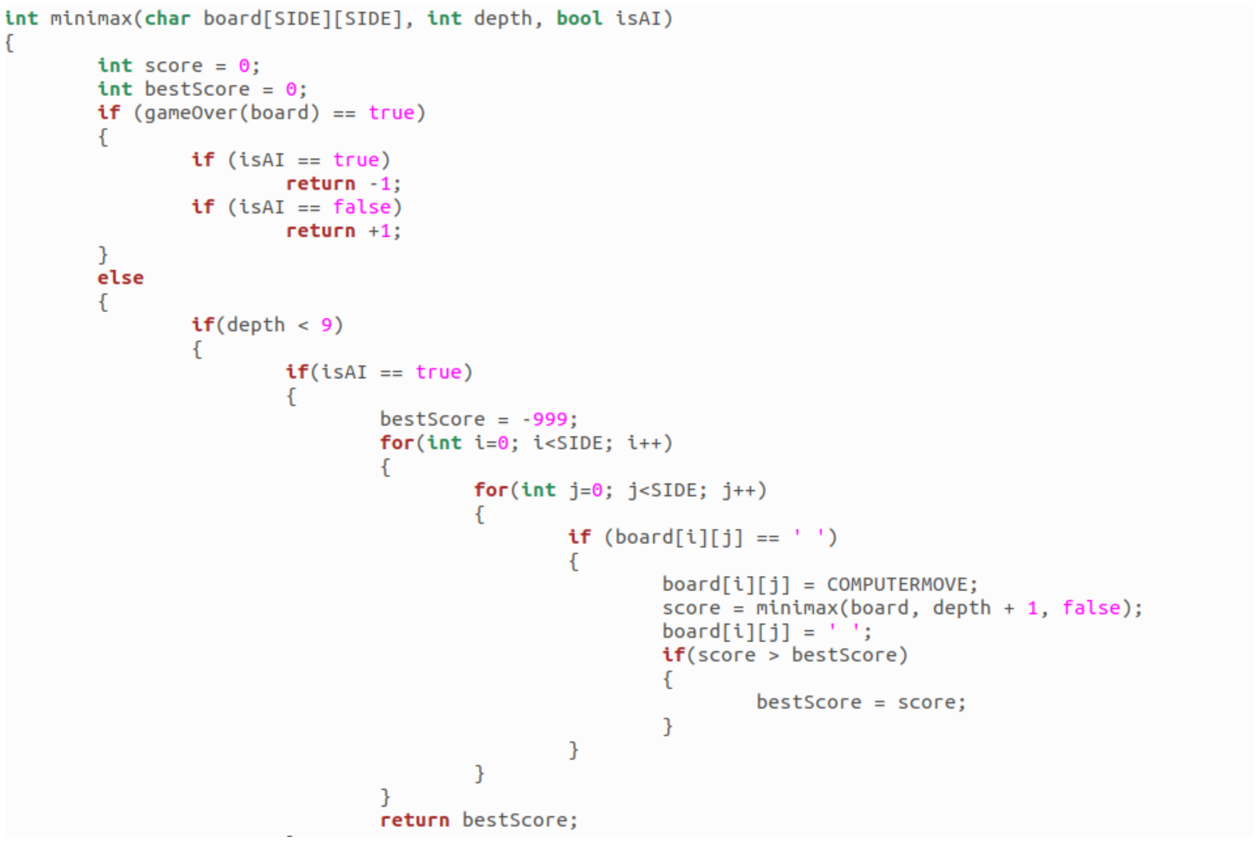


```
int bestMove(char board[SIDE][SIDE], int moveIndex)
{
        int x = -1, y = -1;
        int score = 0, bestScore = -999;
        for (int i = 0; i < SIDE; i++)
        {
                for (int j = 0; j < SIDE; j++)
                {
                        if (board[i][j] == ' ')
                        {
                                board[i][j] = COMPUTERMOVE;
                                score = minimax(board, moveIndex+1, false);
                                board[i][j] = ' ';
                                if(score > bestScore)
                                {
                                        bestScore = score;
                                        x = i;
                                        y = j;
                                }
                        }
                }
        }
        return x*3+y;
}
```

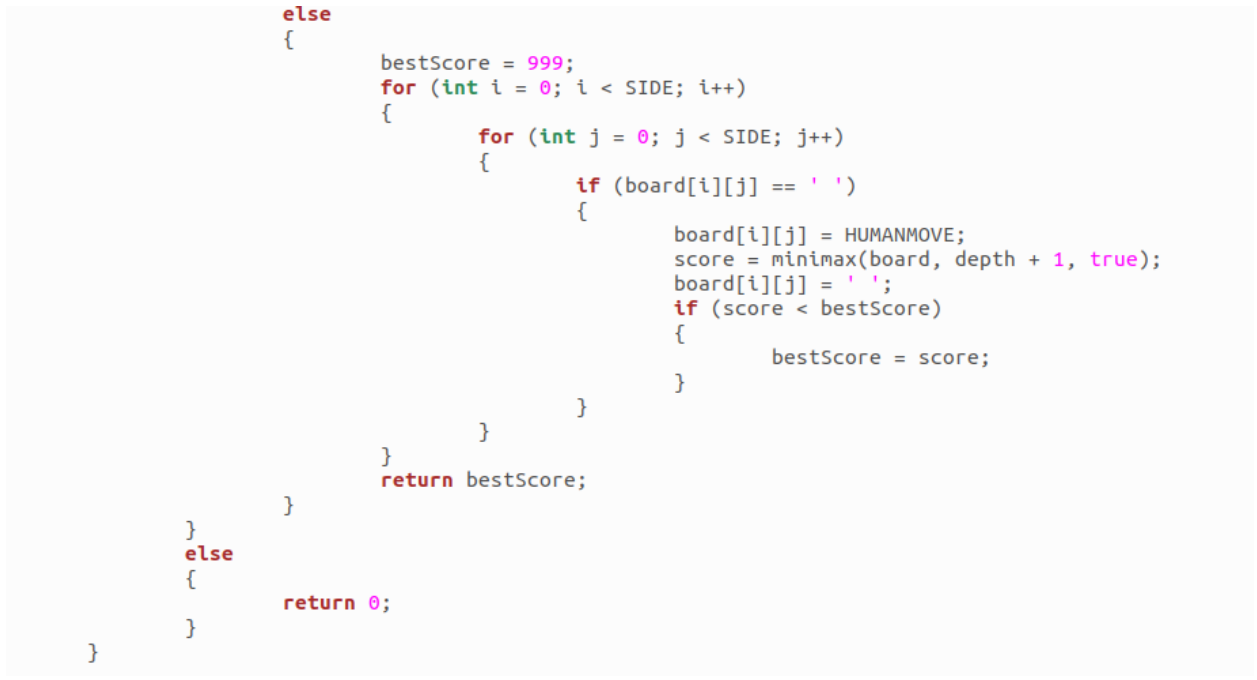
Minimax algorithm

Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing optimally. Its goal is to minimize the maximum loss (minimize the worst case scenario).

minimax() is called to get the bestScore.It takes the arguments board(char),depth(int) and isMaximizingPlayer(bool).If isMaximzingPlayer is true at certain depth,bestScore is assigned to -999, COMPUTERMOVE is assigned to every empty position iteratively score is assigned to recursively called **minimax()** with arguments board,depth+1,false. If score is greater than bestScore , the bestScore is now the score. Return bestScore after all the recursions and iterations.
If isMaximizingPlayer is false,bestScore is assigned to 999,HUMANMOVE is assigned to every possible empty position and recursively calls **minimax()** with arguments board,depth+1,true.If score is less than bestScore , the bestScore is now the score. Return bestScore after all the recursions and iterations.



```
int minimax(char board[SIDE][SIDE], int depth, bool isAI)
{
        int score = 0;
        int bestScore = 0;
        if (gameOver(board) == true)
        {
                if (isAI == true)
                        return -1;
                if (isAI == false)
                        return +1;
        }
        else
        {
                if(depth < 9)
                {
                        if(isAI == true)
                        {
                                bestScore = -999;
                                for(int i=0; i<SIDE; i++)
                                {
                                        for(int j=0; j<SIDE; j++)
                                        {
                                                if (board[i][j] == ' ')
                                                {
                                                        board[i][j] = COMPUTERMOVE;
                                                        score = minimax(board, depth + 1, false);
                                                        board[i][j] = ' ';
                                                        if(score > bestScore)
                                                        {
                                                                bestScore = score;
                                                        }
                                                }
                                        }
                                }
                                return bestScore;
```



```
                        else
                        {
                                bestScore = 999;
                                for (int i = 0; i < SIDE; i++)
                                {
                                        for (int j = 0; j < SIDE; j++)
                                        {
                                                if (board[i][j] == ' ')
                                                {
                                                        board[i][j] = HUMANMOVE;
                                                        score = minimax(board, depth + 1, true);
                                                        board[i][j] = ' ';
                                                        if (score < bestScore)
                                                        {
                                                                bestScore = score;
                                                        }
                                                }
                                        }
                                }
                                return bestScore;
                        }
                }
                else
                {
                        return 0;
                }
        }
```

Conclusion

Tic Tac Toe is a zero-sum and perfect information game. It means that each participant’s gain is equal to the other participants’ losses and we know *everything* about the current game state. The Minimax algorithm is suited for such cases.