Assignment 3

ELEC129 – Introduction to Programming in C

Contents

[Part 1 – Source Code 2](#_Toc32933227)

[Part 2 – Specification 18](#_Toc32933228)

[Part 2 – Analysis 19](#_Toc32933229)

[Part 2 – Design 19](#_Toc32933230)

[Part 2 – Implementation 21](#_Toc32933231)

[Part 2 – Testing and Verification 23](#_Toc32933232)

# Part 1 – Source Code

#include <stdio.h>

#include <stdlib.h>

#define ROW 5

#define COL 5

#define MARKER\_1 'X'

#define MARKER\_2 'O'

#define NO\_MARKER ' '

#define NO\_WINNER 1

#define WINNER\_1 2

#define WINNER\_2 3

#define DRAW 4

void create\_board(char (\*play\_board)[COL]); //Function creates board

int disp\_menu1(); //Displays menu and gets input from user on what mode to play

void play\_multiplayer(char (\*play\_board)[COL]); //Function for when user selects multiplayer

void print\_board(char (\*play\_board)[COL]); //Function to print the board to the user

int check\_board(char (\*play\_board)[COL], int i, int j, int opt2); //Function checks board

void play\_solo(char (\*play\_board)[COL], int opt2); //Function used to pla against the computer

int disp\_menu2(); //Displays menu and gets input from user on if player or computer plays first

int continue\_check(char (\*This\_board)[COL]); //Checks to see if the game is over, who has won and if the game should continue

int disp\_menu3(); //Gets input from user to check if they want to play again

int tactical\_move(char (\*play\_board)[COL], int \*x, int \*y, char marker); //Checks to see if it can make winning move or, if not, block opponent

//Main Function

void main(void)

{

char play\_board[ROW][COL];

int opt1, opt2, game\_on = 1;

//Will do until user enters to not replay

do

{

create\_board(play\_board);

opt1 = disp\_menu1();

//Selects option that user has chosen in previous function

switch(opt1)

{

//Player Vs Player mode

case 1:

printf("\nPlayer Vs Player Mode Selected\n");

play\_multiplayer(play\_board);

break;

//Player vs Computer

case 2:

printf("\nPlayer vs Computer Mode Selected\n");

opt2 = disp\_menu2();

play\_solo(play\_board, opt2);

break;

}

//Checks if user wants to play again

game\_on = disp\_menu3();

}while(game\_on == 1);

}

//Creates the board

void create\_board(char (\*play\_board)[COL])

{

int i, j;

for (i = 0; i < COL; i++)

{

for(j = 0; j < COL; j++)

{

play\_board[i][j] = ' ';

}

}

}

//Menu to select mode

int disp\_menu1()

{

int option = 0;

do

{

printf("MENU:");

printf("\n1 - Multiplayer: Player vs Player");

printf("\n2 - Single Player: Player vs Computer");

printf("\nSelect an option: ");

fflush(stdin);

scanf("%d", &option);

}while(option != 1 && option != 2);

return option;

}

//Plays multiplayer

void play\_multiplayer(char (\*play\_board)[COL])

{

int x, y, game\_over, cont = NO\_WINNER, first, opt2 = 0;

char marker = 'X';

//Prints board after when game begins

print\_board(play\_board);

//Will do until game has concluded

do

{

//Allows players to enter position until they enter valid position

do

{

printf("\nEnter board position x: ");

fflush(stdin);

scanf("%d", &x);

printf("Enter board position y: ");

fflush(stdin);

scanf("%d", &y);

}while(check\_board(play\_board, x, y, opt2));

//Enters chosen move

play\_board[x-1][y-1] = marker;

//Changes which marker is being used to change who is the current player

if(marker == MARKER\_1)

{

printf("\nPlayer 2s turn:\n");

marker = MARKER\_2;

}

else

{

printf("\nPlayer 1s turn:\n");

marker = MARKER\_1;

}

//Prints board when move has been made

print\_board(play\_board);

//Checks to see if game should continue and output to user

switch(continue\_check(play\_board))

{

case NO\_WINNER:

printf("\nThere is no winner yet\n");

break;

case WINNER\_1:

printf("\nPlayer 1 wins\n");

cont = WINNER\_1;

break;

case WINNER\_2:

printf("\nPlayer 2 wins\n");

cont = WINNER\_2;

break;

case DRAW:

printf("\nDraw\n");

cont = DRAW;

break;

}

}while(cont == NO\_WINNER);

}

//Prints board to user with numbers above and to side

void print\_board(char (\*play\_board)[COL])

{

int i, j;

printf("\n\n");

printf(" ");

for (i = 0; i < COL; i++)

{

printf(" %d ", i+1);

}

printf("\n");

for (i = 0; i < COL; i++)

{

printf(" ");

for(j = 0; j < COL; j++)

{

printf("+---");

}

printf("+\n");

printf(" %d ", i + 1);

for(j = 0; j < COL; j++)

{

printf("| %c ", play\_board[i][j]);

}

printf("|\n");

}

printf(" +");

for(j = 0; j < COL; j++)

{

printf("---+");

}

printf("\n");

}

//Checks to see if position entered is valid

int check\_board(char (\*play\_board)[COL], int i, int j, int opt2)

{

//Checks if move is out of bounds

if(i < 1 || j < 1 || i > ROW || j > COL)

{

//Will output only if user has made move and not computer

if(opt2 == 1)

{

printf("\nPOSITION INVALID: Out of Bounds");

}

return -1;

}

//Checks if position is already taken

if(play\_board[i-1][j-1] != ' ')

{

//Will output only if user has made move and not computer

if(opt2 == 1)

{

printf("\nPOSITION INVALID: Already Taken");

}

return -1;

}

return 0;

}

//Play against the computer

//Who plays first depends on previous option

void play\_solo(char (\*play\_board)[COL], int opt2)

{

int x, y, game\_over, cont = NO\_WINNER, first, decision\_made = 0, undecided\_num = 0, a;

char marker = MARKER\_1;

char opp\_marker = MARKER\_2;

printf("\nGame Mode Selected: Single Player\n\n");

print\_board(play\_board);

//Will continue until game has concluded

do

{

//If it is players turn

if(opt2 == 1)

{

do

{

printf("\nEnter board position x: ");

fflush(stdin);

scanf("%d", &x);

printf("Enter board position y: ");

fflush(stdin);

scanf("%d", &y);

}while(check\_board(play\_board, x, y, opt2));

opt2 = 2;

}

//If it is not players turn then it is the computers

else

{

//Do until valid move found

do

{

//Creates a random move for computer

x = rand() % 5;

y = rand() % 5;

x = x +1;

y = y +1;

}while(check\_board(play\_board, x, y, opt2));

//Will replace move if a better tactical move can be made

decision\_made = 0;

decision\_made = tactical\_move(play\_board, &x, &y, marker);

opt2 = 1;

}

//If there is a possible winning move, this will place that marker

//If there is no possible winning move, but it can block opponent from winning, places that marker.

//If there is no possible winning or blocking move, it places a random number

play\_board[x-1][y-1] = marker;

//Changes the marker. This method means that X will always go first regardless of if it is player or computer

if(marker == MARKER\_1)

{

printf("\nPlayer 2s turn:\n");

marker = MARKER\_2;

opp\_marker = MARKER\_1;

}

else

{

printf("\nPlayer 1s turn:\n");

marker = MARKER\_1;

opp\_marker = MARKER\_2;

}

//Prints board

print\_board(play\_board);

//Checks to see if it should continue

switch(continue\_check(play\_board))

{

case NO\_WINNER:

printf("\nThere is no winner\n");

break;

case WINNER\_1:

printf("\nPlayer 1 wins\n");

cont = WINNER\_1;

break;

case WINNER\_2:

printf("\nPlayer 2 wins\n");

cont = WINNER\_2;

break;

case DRAW:

printf("\nDraw\n");

cont = DRAW;

break;

}

}while(cont == NO\_WINNER);

}

//Checks if player or computer should move first

int disp\_menu2()

{

int option = 0;

do

{

printf("Who plays first?");

printf("\n1: Player");

printf("\n2: Computer\n");

fflush(stdin);

scanf("%d", &option);

}while(option != 1 && option != 2);

return option; //returns users option after valid input

}

//Checks to see if game should continue

int continue\_check(char (\*play\_board)[COL])

{

int i, j, marker\_num\_1, marker\_num\_2, empty\_pos;

//Checks for win in rows

for(i=0; i<ROW; i++)

{

marker\_num\_1 = marker\_num\_2 = 0;

for(j=0; j<COL;j++)

{

switch(play\_board[i][j])

{

case MARKER\_1:

marker\_num\_1++;

break;

case MARKER\_2:

marker\_num\_2++;

break;

}

}

if(marker\_num\_1 == COL)

{

return WINNER\_1;

}

if(marker\_num\_2 == COL)

{

return WINNER\_2;

}

}

//CHECKS FOR WIN IN COL

for(j=0; j<ROW; j++)

{

marker\_num\_1 = marker\_num\_2 = 0;

for(i=0; i<COL;i++)

{

switch(play\_board[i][j])

{

case MARKER\_1:

marker\_num\_1++;

break;

case MARKER\_2:

marker\_num\_2++;

break;

}

}

if(marker\_num\_1 == COL)

{

return WINNER\_1;

}

if(marker\_num\_2 == COL)

{

return WINNER\_2;

}

}

//Check diagonal

marker\_num\_1 = marker\_num\_2 = 0;

for(j=0; j<COL; j++)

{

switch(play\_board[j][j])

{

case MARKER\_1:

marker\_num\_1++;

break;

case MARKER\_2:

marker\_num\_2++;

break;

}

}

if(marker\_num\_1 == COL)

{

return WINNER\_1;

}

if(marker\_num\_2 == COL)

{

return WINNER\_2;

}

//Check second diagonal

marker\_num\_1 = marker\_num\_2 = 0;

for(j=0; j<COL; j++)

{

switch(play\_board[j][COL-1-j])

{

case MARKER\_1:

marker\_num\_1++;

break;

case MARKER\_2:

marker\_num\_2++;

break;

}

}

if(marker\_num\_1 == COL)

{

return WINNER\_1;

}

if(marker\_num\_2 == COL)

{

return WINNER\_2;

}

//Check if board is full

empty\_pos = 0;

for(i=0; i<COL; i++)

{

for(j=0; j<COL; j++)

{

if(play\_board[i][j] == ' ')

{

empty\_pos++;

}

}

}

if(empty\_pos == 0)

{

return DRAW;

}

else

{

return NO\_WINNER;

}

}

//Checks if user wants to play again

int disp\_menu3()

{

int option = 0;

do

{

printf("Do you want to play again?");

printf("\n1: Yes");

printf("\n2: No\n");

fflush(stdin);

scanf("%d", &option);

}while(option != 1 && option != 2);

return option;

}

//

int tactical\_move(char (\*play\_board)[COL], int \*x, int \*y, char marker)

{

int i, j, z, marker\_num\_1, empty\_pos, decision\_made = 0;

for(z=0; z<2; z++)

{

//Checks for possible win in rows

for(i=0; i<ROW; i++)

{

marker\_num\_1 = empty\_pos = 0;

for(j=0; j<COL;j++)

{

if(play\_board[i][j] == marker)

{

marker\_num\_1++;

}

if(play\_board[i][j] == NO\_MARKER)

{

empty\_pos++;

}

}

if(marker\_num\_1 == COL-1 && empty\_pos == 1)

{

\*x = i+1;

for(j=0; j<COL; j++)

{

if(play\_board[i][j] == NO\_MARKER)

{

\*y = j+1;

decision\_made = 1;

return decision\_made;

}

}

}

}

//Checks for possible win in columns

for(j=0; j<ROW; j++)

{

marker\_num\_1 = empty\_pos = 0;

for(i=0; i<COL;i++)

{

if(play\_board[i][j] == marker)

{

marker\_num\_1++;

}

if(play\_board[i][j] == NO\_MARKER)

{

empty\_pos++;

}

}

if(marker\_num\_1 == COL-1 && empty\_pos == 1)

{

\*y = j+1;

for(i=0; i<COL; i++)

{

if(play\_board[i][j] == NO\_MARKER)

{

\*x = i+1;

decision\_made = 1;

return 1;

}

}

}

}

//Check diagonal

marker\_num\_1 = empty\_pos = 0;

for(j=0; j<COL; j++)

{

if(play\_board[j][j] == marker)

{

marker\_num\_1++;

}

if(play\_board[j][j] == NO\_MARKER)

{

empty\_pos++;

}

}

if(marker\_num\_1 == COL-1 && empty\_pos == 1)

{

for(j=0; j<COL; j++)

{

if(play\_board[j][j] == NO\_MARKER)

{

\*x = j + 1;

\*y = j + 1;

}

}

}

//Check second diagonal

marker\_num\_1 = empty\_pos = 0;

for(j=0; j<COL; j++)

{

if(play\_board[j][COL-j-1] == marker)

{

marker\_num\_1++;

}

if(play\_board[j][COL-j-1] == NO\_MARKER)

{

empty\_pos++;

}

}

if(marker\_num\_1 == COL-1 && empty\_pos == 1)

{

for(j=0; j<COL; j++)

{

if(play\_board[j][COL-1-j] == NO\_MARKER)

{

\*x = j + 1;

\*y = COL-j;

}

}

}

//Changes around what marker is being used for blocking move. This way computer can either be 'X' or 'O'

if(marker == MARKER\_1)

{

marker = MARKER\_2;

}

else

{

marker = MARKER\_1;

}

}

return decision\_made;

}

# Part 2 – Specification

The objective is to create a program, utilizing the C language, that implements a tic-tac-toe game. The program should play on a 5X5 board, although the program should intentionally be designed in a way that permits the ability to easily change the number or rows and columns used with only a small quantity or reprogramming necessary.

The program should include two sperate playable modes. The program should prompt the user for them to select the option of which of these game modes that they want to play. If multiplayer (player vs player) is selected, then the program should progress to start the game. If single player (player vs computer) is selected, then it should also ask who plays first. If an unacceptable answer is entered it should request new input. This is to ensure that the program is more robust.

The first player, regardless of whether they are human or computer, should always play as “X” and the second player should always be “O”. Both players should only ever be able to place markers on an empty position. If a taken position is entered, then the game should not place that marker and request for a different position to be entered. The game should continue playing until the game is either won by a player or if there are no more possible moves possible. The result of the game should also be printed to the user. The objective for the player is to place 5 markers in a line and the game should be able to recognize that this event has occurred.

Initially, the program is required to print this board in a neat and organized format to the user. It should also include numbers above and alongside the grid to simplify the use and avoid confusion on how to correctly enter coordinates. It should also print after every move.

The “player vs computer” version should play similar to the “player vs player”. However, opponent should be played by a computer. It should make a random move unless it can win or block the opponent from winning. It should prioritise winning rather than blocking.

When a game has concluded, it should ask they user if they want to play again. If so then the program should repeat and reset the board.

# Part 2 – Analysis

Inputs:

* Game mode option (integer)
* Player first or computer first option (integer)
* X coordinates (integer)
* Y coordinates (integer)
* Play again option (integer)

Outputs:

* The play board of markers (char)
* The “O” marker (char)
* The “X” marker (char)

Other relevant aspects:

* Display to the user the result of the game (if player 1 won, player 2 won, draw, or if game is continuing)

Constraints

* When returning coordinates the program will need to return multiple integers.

# Part 2 – Design

This section includes the algorithm used to design the program along with explanations of the steps.

Step 1

While loop used at start to contain code so that the game will be repeated until the user specifically enters that they do not want to play again.

Step 2

1. Function that creates board of empty ‘ ’ chars in 5X5 grid (based on definition so this can be easily changed to give different sized grid with program still working).
   1. For every row, until it has completed all columns, it repeats.
      1. For every row, until it has completed all columns, it repeats
         1. This combination of 2 for statements completes the 5X5 grid, filling it with an char that shows that position is empty and not yet taken by opponent.

Step 3

1. Function that displays menu to user. Gets input from user before returning it to program concerning what game mode to play
   1. While used so that this will repeat until acceptable input as been entered. This makes the program more robust.
      1. Prints instructions to user
      2. Gets input
   2. Valid input is returned.

Step 4

1. Switch is used to determine which to use based on the output of previous function
   1. First case: “player vs player”
      1. Prints notice to user on selected option
      2. Function to play against another player
         1. Prints board to begin
         2. While there is no winner repeat this section of code
            1. While the user has not yet input valid input (decided using a function to check if it out of bounds or if the place has already been taken)

Print instructions to user

Get input for x

Get input for y

* + - * 1. Return valid input
      1. Enter move into board, changing empty marker to which ever players marker is being used
      2. If the marker is “X” change it to “O”, if it is “O” change it to “X” (This swaps the current player whilst always ensuring that “X” plays first)
      3. Print board after move has been made
      4. Use a switch along with a function to check the board to see if game should continue. Output this result to user.
  1. Second case: “Player vs Computer”
     1. Prints notice to user on selected option
     2. Function to get input from user on if player or computer should play first.
        1. While used so that this will repeat until acceptable input as been entered. This makes the program more robust.
           1. Prints instructions to user
           2. Gets input
        2. Valid input is returned.
     3. Function to play the game against computer
        1. Prints board to begin
        2. If it is the players turn
           1. While there is no winner repeat this section of code

While the user has not yet input valid input (decided using a function to check if it out of bounds or if the place has already been taken)

Print instructions to user

Get input for x

Get input for y

Return valid input

Change who’s turn it is for the next round

* + - 1. Else it is the computers turn
         1. Make two random numbers until a compatible combination is found.
         2. Check to see if a better move can be made. Will return integers if it finds better move.

Will go through this section twice. First using own marker to find win and then using opponent’s marker to find way to prevent them winning. If it does at any point, then it will stop.

Checks for win in rows

Checks for win in columns

Checks for win in diagonal

Checks for win in the other diagonal

Uses if and else statement to change around marker

Return decision made

* + - 1. Enter move into board, changing empty marker to which ever players marker is being used
      2. If the marker is “X” change it to “O”, if it is “O” change it to “X” (This swaps the current player whilst always ensuring that “X” plays first)
      3. Print board after move has been made
      4. Use a switch along with a function to check the board to see if game should continue. Output this result to user.

Step 5

1. Function used to get input from user regarding if they want to play again (If not this will stop the while function from repeating the program. This will result in the program ending.)

End of algorithm

# Part 2 – Implementation

Function – create\_board(char (\*play\_board)[COL])

The purpose of this function is to create the board required to play the game. When it does this, it creates the board completely blank so that it can output for use in play. The function is called near the start of every instance of the while function used to repeat the game. The intention of this is so that it will reset the board each time the user wants to play.

It works by utilizing the “for” twice. This combination allows the function to run through each line that it has to create for the column and then place a char in each position of the rows to mark that the position is empty.

Function – disp\_menu1()

The menu is displayed to find what game mode the user wants to play.

This function outputs instructions to the user using printf several times. It uses fflush(stdin) before asking for input from the user. This code has been included within a while so that will only stop when the option entered by the user is valid. This is to make the program more robust. When this has completed, the option the user has entered is returned so that the game mode can be selected.

Void play\_multiplayer(char (\*play\_board)[COL])

First, it calls another function to print the board as it currently is. Then it repeats the next section until game has finished. It asks for user input until they have input acceptable coordinates and then uses this to edit the play\_board. It then changes the player marker for the opponent. A switch is used to determine if the game should continue

Int disp\_menu2()

Prints menu to user and finds if they want the computer or the player to go first when playing against the computer. Gets option and returns to function.

int continue\_check(char (\*play\_board)[COL])

Checks for if player 1 or player 2 has won, the game is a draw, or if it should continue. Returns result to the main function. It checks the rows, columns and both diagonals

Int disp\_menu3()

Outputs instructions to the user, finds if they want to play the game again and returns this choice.

int tactical\_move(char (\*play\_board)[COL], int \*x, int \*y, char marker)

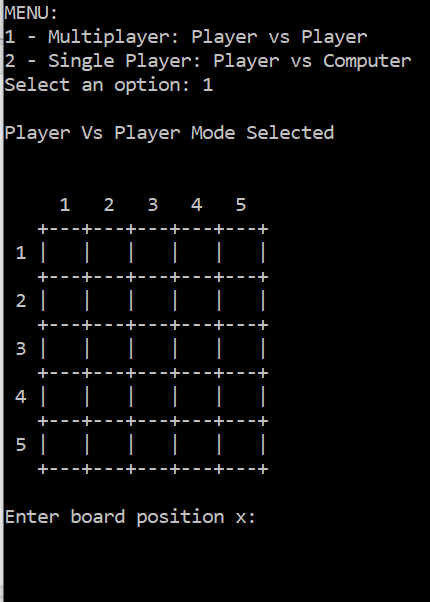
This function uses a for so that it will run though the code twice, or until it finds a position to take. It first checks to see if it is cable of winning in any of the rows, columns or diagonals. It does this by taking in the current marker and using this to determine which one it is playing with. The function then checks to see if any lines have four of its own markers and 1 empty marker. If it does then it runs through that line again to find where the empty position and returns the coordinates to the main function.

If it can not locate any possible win then it will repeat this code however, the marker that it has been searching for will be replaced by the opponent’s marker. If it finds that it will treat it the same way and use that as the position that it wants to take, thus preventing that way of winning. This means that the only possible way a user should be able to win is by simultaneous having two possible methods of winning become available at the same time.

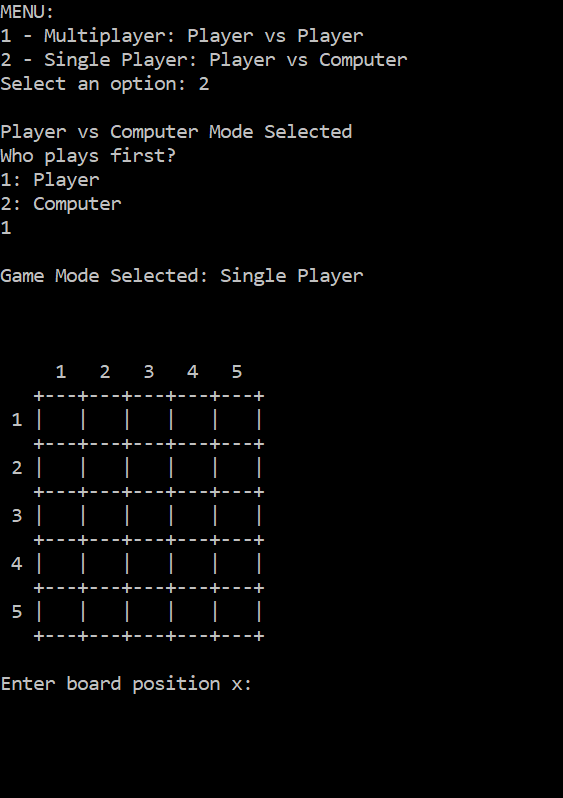
# Part 2 – Testing and Verification

For the purposes of testing, an additional section of code has been implemented to create a scenario best suited to check if the program is working correctly. The following code start the program with four “X” markers and 1 blank to see if the program will be capable of detecting this and filling in the line. This section of code is not in the final code. However, the first section will test the menu and after will show a game as it is meant to be played and a player vs player game.

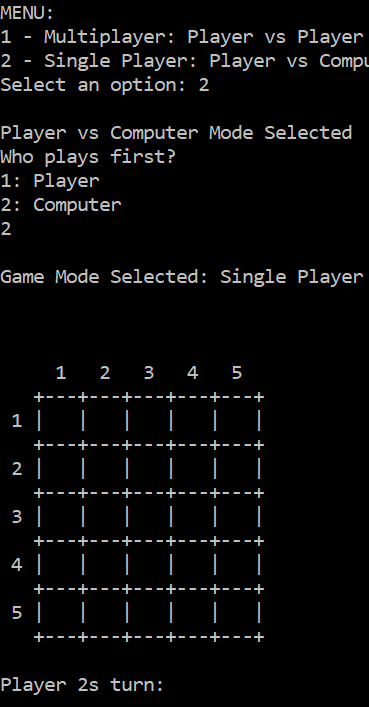
Menu Test

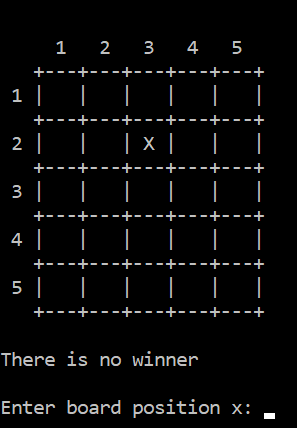


User can select player vs player and it works.

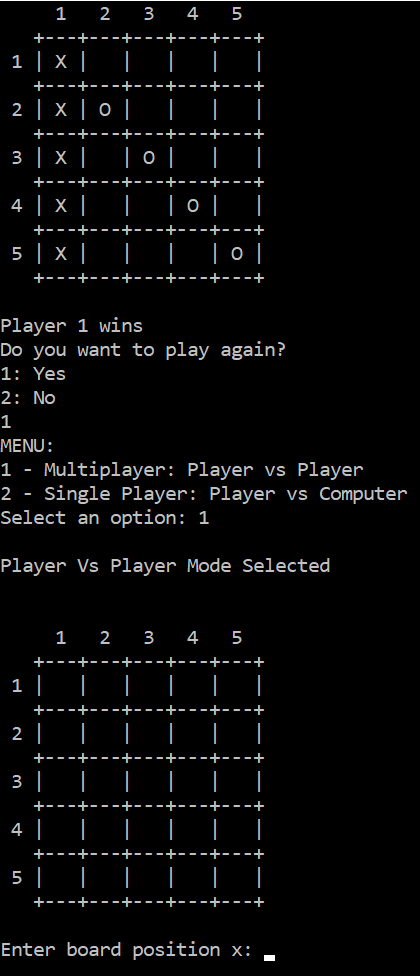


Player can select player vs computer and go first.



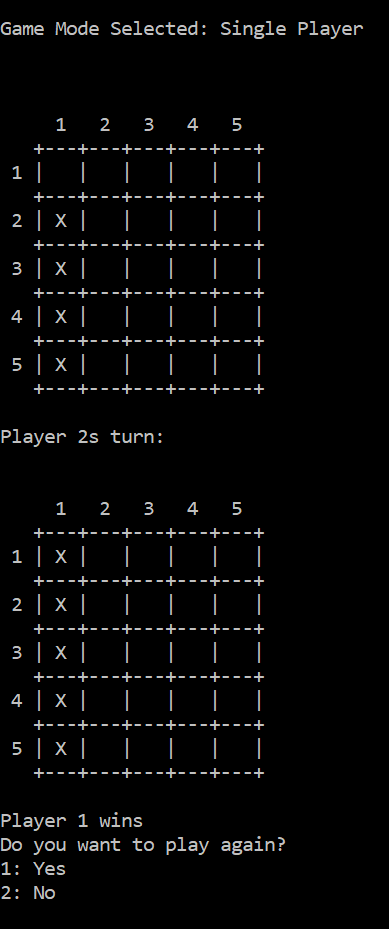


Player can select player vs computer and have computer go first.

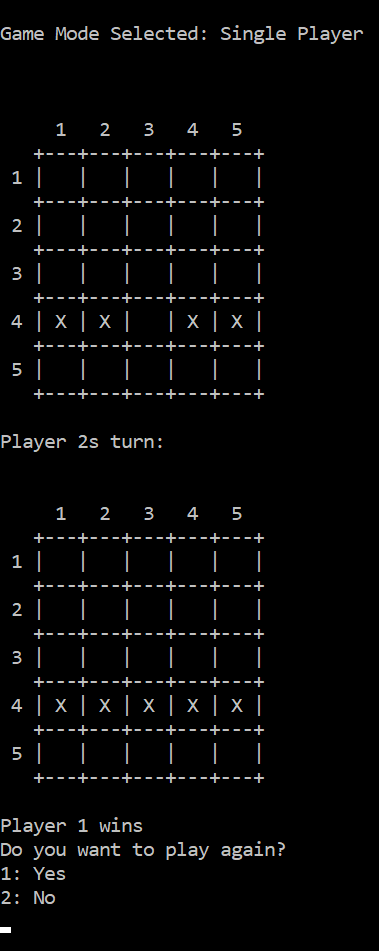


Player can play another game and the board will reset.

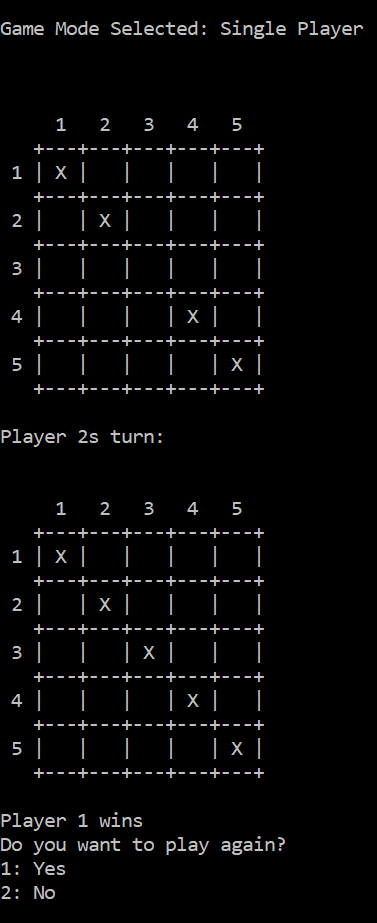
Winning Move



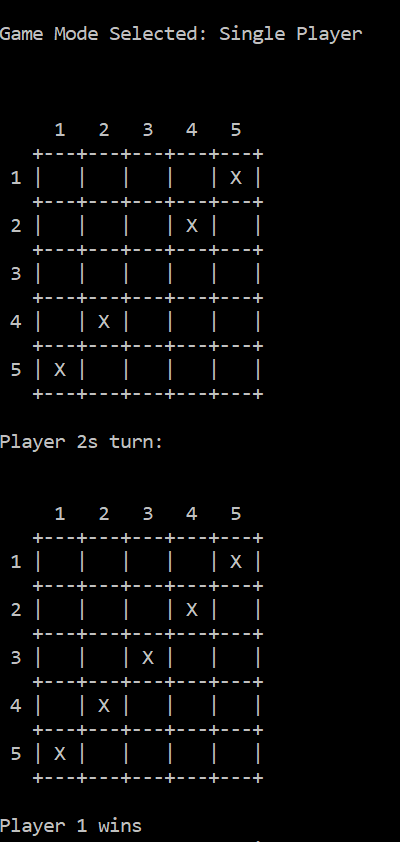
Computer can win in column. This test is successful



This test is successful. Computer was able to spot and select win in row



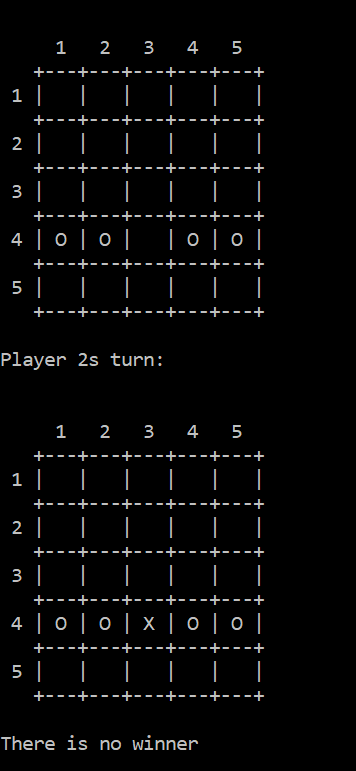
The first diagonal test worked correctly as it was able to see the possible win and take it.



The second diagonal test was successful.

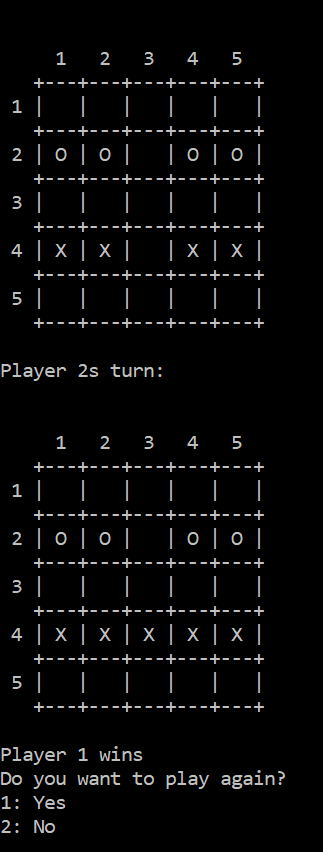
Blocking Move

The computer will now attempt to block the opponent from winning

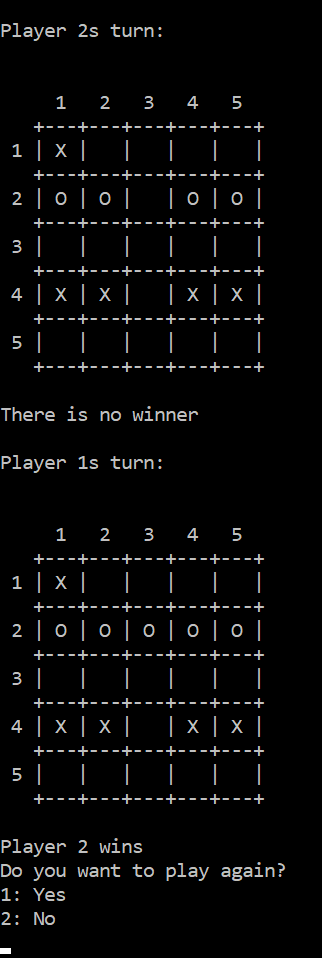


This test is also successful. The computer was able to block the player from being able to win.

Prioritising

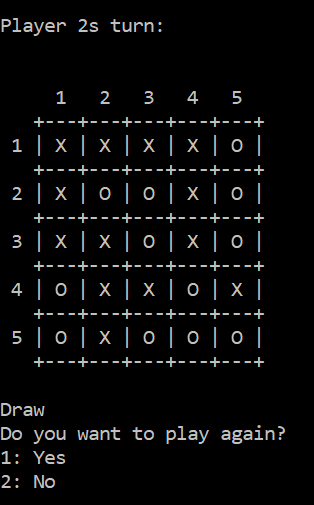


Computer can successfully prioritise winning over blocking the opposing user.



In this test, the computer plays as the “O” and the same scenario plays out as last time to see if it will now prioritise the other line. A position of x = 1 and y = 1 has been entered from the user to move along the program. This also shows that the game works playing as the other marker.

Playing normal



Game plays as intended. The computer behaved as expected.

Conclusion

It appears that all test cases work exactly as expect and the computer behaves as intended. This suggests that the program works correctly and fulfils the specification.