TIC TAC TOE GAME

CIS 37 Final Project

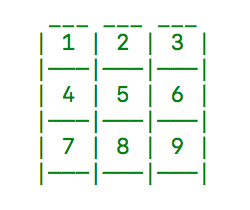
Hoang Nguyen, Triet Ho, Zhuoxuan Wang, Jack Wang

1. **Introduction:**

Since the final project is an opportunity to apply what we have learnt and to test our programming skills, we come up with the idea to write a game which can cover many aspects of C such as array, function, strings, loops…etc. After discussing and doing research, we decided to write a simple tic tac toe game which considered to be simple, fun, and interactive.

1. **About the project:**
2. Overview:
   * Tic tac toe is an easy-playing game designed for children to develops abilities such as strategy and corporation skills.
   * Rules: It begins by first player place their mark in the 3x3 square and then the other will do the same thing. The game keeps going until one player have three in row either vertical, horizontal or diagonal. Thus, that player will win the game. The game is draw when the board is all filled but none of players win.
   * The outcomes of the program are to provide a user interface environment throughout the console, where two players can play against each other. The players will enter input from keyboard turn by turn until one win. The player can also play against computer.
3. How to transfer the ideas into programming:

* First, define an integer two-dimensional array and represents it as 3x3 grid. Define the value to assign inside the array, which X has the value of 1, O has the value of 2 and blank space has the value of zero.
* Declare a loop that keep the game playing until one win or draw
* Then, using functions to assign the value inside the array. The function will require the user to enter a specific number equivalents to the number of the square from left to right, up to down (Multiplayer). Moreover, there will be also a function that automatically assign value in the array which is used to play against the player (AI versus).



The number equivalent to the number of square

* To make each frame transfer smoothly, use clear screen function and then print a new board again, which this time updated new moves from player
* Check to see if one side win. If it true end the loop and print out the result.
* Else, the next player will play
* Continue to check
* Update the board

1. **Project design**
2. Main Function: In main function, the idea of the game is first print a blank board. Then, take input from player and check to see if player has won. Otherwise, take input from player 2/ Computer and continue to check. During the game, count steps to see if after nine steps ( board filled ) and no one has won, the game draw. Redraw the board each turn to update.

int main()

{

    int board[SIZE][SIZE]={{0,0,0},{0,0,0},{0,0,0}};

//declare a board with all 0 as blank spaces

    DrawBoard(board);

    while(game==CONTINUE)

    {

        Player1(board);

        ++count; //Count the steps to check draw game

        DrawBoard(board);

        if(Check(board)!=1&&game==CONTINUE)//If player 1 not win

        {

            //Player2(board);       //Multiplayer

            Computer(board);        //Computer versus

            ++count;                //Count step

            DrawBoard(board);

            Check(board);        //Check to see if player 2 win

        }

    }

    return 0;

}

1. Functions prototype:

void DrawBoard(int board[][SIZE]);           //function draw board

int Check (int board[][SIZE]);        //Return 1 if one player has 3 in row. Otherwise, return 0

void Player1(int board[][SIZE]);    //Player 1 assign on the board

void Player2(int board[][SIZE]);    //Player 2 assign on the board

void Computer(int board[][SIZE]);   //AI computer assign on the board

int LineCheck(int board[][SIZE]);   //Used by Check

int DiagonalCheck(int board[][SIZE]);   //Used by Check

int Defense(int playerInput[], int length  ,int board[][SIZE]);    // Used by Computer

int Attack(int board[][SIZE]);  //Used by Computer

1. Global Variables:

#define SIZE 3

struct Pair         //Used by Computer AI to find pair

{

    short num1,num2;

}pair;

int count=0;        //count turns

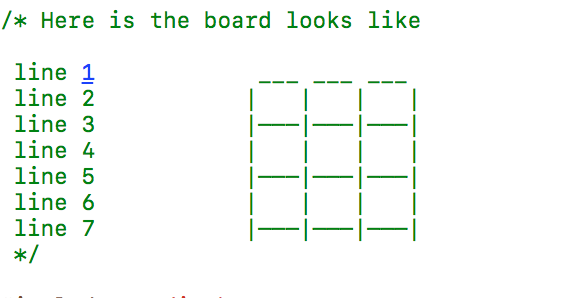
int playerinput[5]={0};//store previous positions of player for computer to process

unsigned lengthOfPlayerInput=0;

enum Game\_Status{END,CONTINUE,DRAW};    // define game status

enum Game\_Status game=CONTINUE;      //game status to store result

1. Functions:
2. Draw board



void DrawBoard(int board[][SIZE])

{

    system("clear"); // Clear screen each time called

    puts("\*\*\*\*\*GAME STARTS\*\*\*\*\*\n");

    puts("Two players, X and O, who take turns marking the spaces in a 3×3 grid.\n"

         "The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.");

    for(int i=0;i<SIZE;++i)

    {

        if(i==0)

        {

            printf("%\*c",40,' ');       //print 40 character spaces

            printf("\_\_\_ \_\_\_ \_\_\_\n");    //draw line 1

        }

        printf("%\*c",38,' ');           //print 38 character spaces

        for(int j=0;j<SIZE;++j)

        {

            printf(" | ");

            switch(board[i][j])

            {

                case 1: printf("x");

                    break;

                case 2: printf("o");

                    break;

                default: printf(" ");

                    break;

            };

        }

        printf(" |");                   //close line 2,4,6

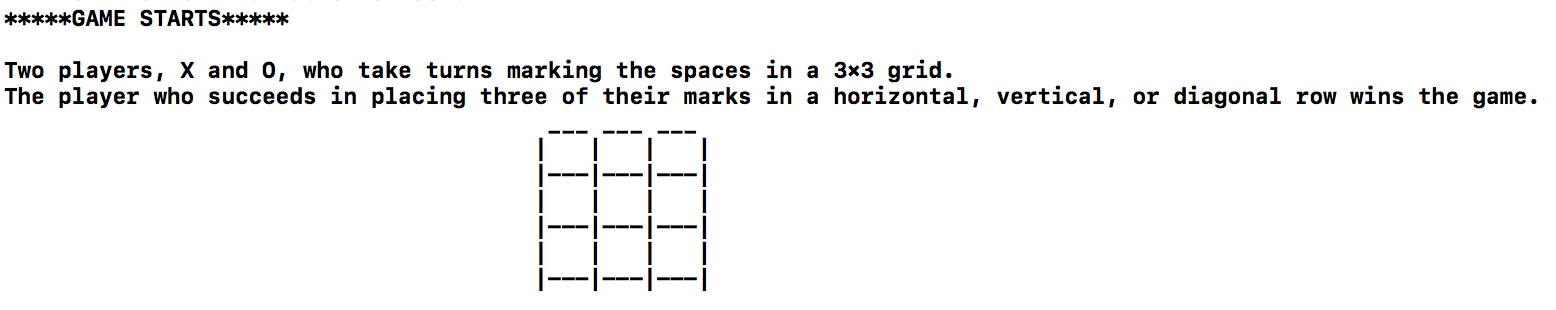
        printf("\n%\*c",39,' ');   //go to new line and print 39 character spaces

        printf("|———|———|———|\n");        // draw line 3,5,7

    }

    printf("\n");

}



1. Player Function: create a temporary variable to store the input from player. Check that number to see if the place was already selected or not. If the position was selected, let the player know and ask them to enter again. One thing special is inside the function, two global variables playerinput[5] and lengthOfPlayerInput are used to save the player input. Then the computer will know what moves player had placed to defend against player.

void Player1(int board[][SIZE])

{

    int x;

    do{

        printf("The first player 'X' please choose a position (1-9) :");

        scanf("%d",&x);

        switch(x)

        {

            case 1:

            case 2:

            case 3:

            case 4:

            case 5:

            case 6:

            case 7:

            case 8:

            case 9:

                if(board[(x-1)/SIZE][(x-1)%SIZE]==0) //If a player enter a position that is empty

                {

                    board[(x-1)/SIZE][(x-1)%SIZE]=1;

                }

                else   //If a player enter a number that already placed

                {

                    printf("It already placed\n");

                    x=-1;            //assign x to -1 to continue the loop

                }

                break;

            default:

                printf("Invalid Input\n");    // If the player enter a number not from 1-9

        };

    }

    while(x<1||x>9);

    playerinput[lengthOfPlayerInput]=x;     //store input of player for computer to process

    lengthOfPlayerInput++;      //know the length of array input

}

Although the player only input the number from 1-9, we need to convert that number into the number of row and column to access the address of the two-dimensional array. The solution is here:

Assume x is the number entered by player and

Board[row][column] is the address we want to access, then:

row=(x-1)/(the number of columns)

column=(x-1)%(the number of rows)

            0   1   2

1=1 + 0\*3 + 0

2=1 + 0\*3 + 1

3=1 + 0\*3 + 2

4=1 + 1\*3 + 0

…..

x= 1+ i\*3+ j

which i: the index of row

3: the number of columns

j: the index of the array

            \_\_\_ \_\_\_ \_\_\_

      0   | 1 | 2 | 3 |

          |———|———|———|

      1   | 4 | 5 | 6 |

          |———|———|———|

       2   | 7 | 8 | 9 |

           |———|———|———|

1. Checking Function: return 1 if one player has three marks in row. There are two sub-functions inside the check. Specifically, the line check will return non-zero value, which is 1 for player X and 2 for player O, if it found three values in row either in horizontal or vertical. Similarly, the DiagonalCheck will checks and return as diagonal.

int Check(int board[][SIZE])

{

    int result=(LineCheck(board)+DiagonalCheck(board));

    if(result)

    {

        if(result == 1)

            printf("Player X win the game\n");

        else

            printf("Player O win the game\n");

        game=END;

    }

    if(count==9&&result!=1)

    {

        printf("It is a draw game\n");

        game=DRAW;

    }

    return result;

}

the result take the input of two sub-function, which is LineCheck and DiagonalCheck. At first we intend to use the result with the condition operator OR

int result=(LineCheck(board)||DiagonalCheck(board));

However, we won’t be able to know which player had won because the LineCheck and DiagonalCheck all return non-zero value if it they found three in row. Therefore, we add the returned values together to know if it either 1 or 2.

int result=(LineCheck(board)+DiagonalCheck(board));

int DiagonalCheck(int board [][SIZE])

{

    //return 1 if player X has 3 in the diagonal

    //return 2 if player O has 3 in the diagonal

    if (((board[0][0] == 1) && (board[1][1] == 1)&&(board[2][2]==1))||

        ((board[0][2] == 1) && (board[1][1] == 1)&&(board[2][0]==1)))

    {

        game=END;

        return 1;

    }

    if (((board[0][0] == 2) && (board[1][1] == 2)&&(board[2][2]==2))||

        ((board[0][2] == 2) && (board[1][1] == 2)&&(board[2][0]==2)))

    {

        game=END;

        return 2;

    }

    //return 0 if none of players win

    return 0;

}

int LineCheck(int board[][SIZE])

{

    for(int i=0; i<SIZE; ++i)

    {

        if((board[i][0]==1 && board[i][1]==1 && board[i][2]==1) || (board[0][i]==1 && board[1][i] == 1 && board [2][i]==1))

        {

            game = END;

            return 1;

        }

        if((board[i][0]==2 && board[i][1]==2 && board[i][2]==2) || (board[0][i]==2 && board[1][i] == 2 && board [2][i]==2))

        {

            game = END;

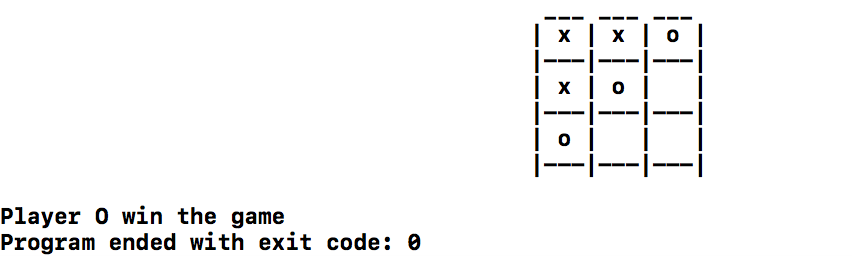
            return 2;

        }

    }

    return 0;

}



The Computer Function: The priority of this function is to win player. Therefore, it checks to see if it can win. If not, it will check to see if it need to defend the player’s moves. If two previous conditions were not satisfied, then it just place a random position inside the grid. Because the attack and defend function have the similar task, we only introduce one.

void Computer(int board[][SIZE])

{

    srand((unsigned)time(NULL));

    int tmp;

    if(Attack(board)==0){   //Attack

        if(Defense(playerinput,lengthOfPlayerInput,board)==0){  //Defense

            //Place a random position

            do

            {

                tmp=1+rand()%9;         // random one number from 1-9

            }while(board[(tmp-1)/SIZE][(tmp-1)%SIZE]==1||board[(tmp-1)/SIZE][(tmp-1)%SIZE]==2);  // keep random if the number has already been chosen

            board[(tmp-1)/SIZE][(tmp-1)%SIZE]=2;// computer place 'O' from chosen number

        }

    }

}

Attack Function: The goal of attack function is to find inside the grid the line that already has two marks and one blank space. Go into that line, find the index where blank space is and assign the value of the mark to that position. Using the loop to check all possibilities

int Attack(int board[][SIZE])

{

    for(int i=0;i<SIZE;++i)

    {

        //row check

        if(sin(board[i][0])+sin(board[i][1])+sin(board[i][2])==sin(0)+sin(2)+sin(2))

            for(int j=0;j<SIZE;++j)

                if(board[i][j]!=2){

                    board[i][j]=2;

                    return 1;

                }

        //column check

        if(sin(board[0][i])+sin(board[1][i])+sin(board[2][i])==sin(0)+sin(2)+sin(2))

            for(int j=0;j<SIZE;++j)

                if(board[j][i]!=2){

                    board[j][i]=2;

                    return 1;}

        //diagonal check

        if(sin(board[0][0])+sin(board[1][1])+sin(board[2][2])==sin(0)+sin(2)+sin(2))

            for(int j=0;j<SIZE;++j)

                if(board[j][j]!=2){

                    board[j][j]=2;

                    return 1;

                }

        if(sin(board[0][2])+sin(board[1][1])+sin(board[2][0])==sin(0)+sin(2)+sin(2))

            for(int j=0;j<SIZE;++j)

                if(board[j][SIZE-(j+1)]!=2){

                    board[j][SIZE-(j+1)]=2;

                    return 1;

                }

    }

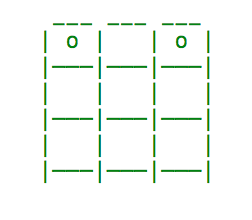
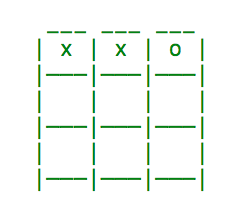
    return 0;

}

At first, we add the sum of all index in that line to find the specific condition. In this example, because two O have each value of 2 and then add the value of blank space with equal to zero, the sum will be 4.

 board[i][0]+board[i][1]+board[i][2]==4

However, the program will have the same result in these two circumstances because both sums are all equal 4

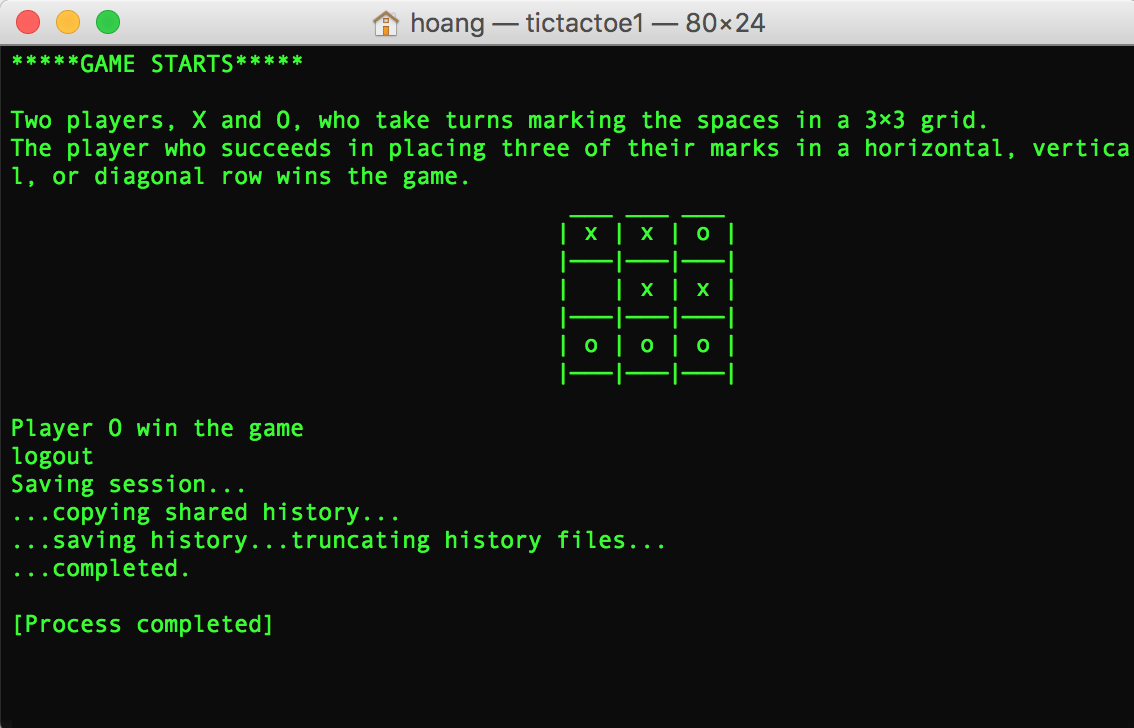


To solve this, we try many formulas but all leads to the same result with another circumstance. Therefore, we make it become unique by using the geometric sin(x)

sin(board[i][0])+sin(board[i][1])+sin(board[i][2])

==sin(0)+sin(2)+sin(2)

1. Testing Result



Final result

1. Problems and Solutions ( What have learned)
2. Problems:

* Some conditions seem to be true but can lead to unexpected actions
* Sometimes, need to change the value of variable but that variable is used widely
* The main function is complicated and hard to understand
* Changing one code can change the program
* Forget what doing with the program

1. Solutions:

* Check carefully the condition by testing many inputs and see if it corrects for all.
* If variables are constant and used frequently during program, it will be better to define them before using.
* Try to reduce the main function by divided into many functions which have specific purposes. Also, clearly naming is crucial to know what the meaning of variables or functions. Write a function prototypes first to know brainstorm how the main function works.
* Again, if a specific task is used repeatedly, change it into a function so that the programmer can reuse. Moreover, test each function before using all of them to avoid unknown errors. If errors exist, debugging may help.
* Comment is important when working with teammates

1. **Summary:**

When working on the project, we learnt how to create a game using C. However, some parts of the code are tedious and hard to understand. For example, there are still unnecessary variables such as or loops that may reduce the speed and the memory of the program. In the future, the game can be improved by adding some features such as menu screen for the player to select the mode and some improved functions.

1. **Reference:**

<http://www.cprogrammingnotes.com/question/tic-tac-toe-game.html>