2019

Algorithms and Data Structures

SET08122

FICO, FRANCESCO (40404272)

Contents

Introduction

The problem that has been given to tackle as coursework for the Algorithms and Data Structures course (SET08122) is of design and develop the game of Tic Tac Toe (Noughts and Crosses). The game must be developed in the C programming language, compiled and played from the command line; must be a complete game to play, with a board, two players and consequent winning conditions.

Design

The beginning of the designing phase started on paper, literally playing the game a dozen times, to better understand the dynamic on a programming logic perspective. This kind of game can seem trivial at the beginning but contains a lot of tricky choices that the developer must make under the surface; the first of which is how to design, implement and draw the actual board for the game.

Having a fixed amount of element for this classic game (9 positions for the board), the immediate choice that comes in mind on where to store the players pieces is an array of strings in which the players can position their pieces choosing a number between 1 and 9, easily implementing the three columns drawing the array in three separate lines. Even if this seemed the most logical choice, after the first lines of code, it was very clear that using this approach would have been tricky to debug later in the development phase. It did not take a while to understand that a two-dimensional array was the way to go; basically, multi dimension arrays are arrays inside an array, this approach seemed perfect for the creation of the columns and rows for our game board, easier to understand and easier to debug.

After creating, initializing and drawing the game board on screen, was time to design in which way accept input from the players. This particular function was easy enough to design and implement, even if C is a fairly new and obscure language for me; the player has to input two integers that resemble the coordinates for the game board (columns and rows), the function will translate the position given by the player in the actual positions stored in the array (e.g. the position 1,1 for the player is the actual 0,0 for the array) and then it checks if the position chosen is actually free (at the creation of the game, all the positions in the array has been filled with blank spaces to better check the winning condition in another function later on), only after that, if the validation has been passed, the function writes the player move in the array.

For the second player, for testing and debugging purposes, it was just logical to adapt the same function from the first player, changing the character on the board from a cross to a nought. From a designing perspective this was a decision dictated by momentary convenience, just to have a playable game to test for later design a sort of artificial intelligence

to let the user play against the computer itself. With the two players in place and all the pieces on the board, it was time to set the rule for the actual game itself.

The winning condition for the game is straight forward, the first player to have three pieces in a line win. The function responsible to checking the winning conditions in the game will check first the columns, then rows and in the end the diagonal lines for pieces aligned together and, if found, will return the first position checked; otherwise it will return a blank space. With the game in place and working I tried to implement an automatic player to play against.

The first working computer player it is a function that checks for the first available empty space in the board and writes a nought in it. This function also declares a game draw if there are no spaces left and no winning condition is met. As explained, this function is simple and not very fun to play against. For this reason, I started developing a more heuristic function with a couple more rules in it, the artificial player will check available space in the four corners and start there, otherwise it will check the centre of the board and, in the end, it will check for any additional empty blocks and write the nought in one of them. This function also contains the same condition for a draw game from the latter simple AI function.

Enhancements

Unfortunately, I was not able to implement the additional functionalities requested from the coursework. However, if had more time, I have ideas of how those functionalities could have been implemented, for instance the use of a stack to store the players movements on the board to have the functionality to remove and add the last item inserted in the stack (Least In - First Out) for the function of undo /redo; or the use of a Min/Max algorithm to create an artificial player that was able to play a perfect game based on the choices of the player one (Surma 2018).

Critical Evaluation

Setting aside personal dissatisfaction with not having been able to implement additional functionalities, the sentiment toward the design and implementation for the program is positive. The only functionality that is behaving poorly is the heuristic one for the computer player, unfortunately when the player chooses to place the cross in one specific corner (2,2), the function malfunctions and does not follow all the necessary checks to continue to play. After debugging for a while, I was not able to understand the reason why this specifically behaviour. Inside the source code there are all the commented parts to play the game with a second human player, a simple AI or the heuristic one.

Personal Evaluation

This was the first time I had the possibility to work on the design and the implementation of a game, plus was my first time using the C programming language (if not considering the course labs). Never, programming in java or C# in my learning experience, I had to consider the various data structures and primitives to better use and optimise the program I was designing. Unfortunately, I do not feel like I took advantage from all the knowledge that the lectures and the labs gave me, focusing too much of the playability of the game without appropriately applying the necessary data structure and algorithms to the problem given.

References

Surma, G., 2018. *Tic Tac Toe – Creating Unbeatable AI* [online]. Towards Data Science. Available from: https://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7d [Accessed 20/03/2019]

Appendix

```c
#include <stdio.h>


///-----------------------------------------------------------------
///   Program:        TicTacToe (C Program)
///
///   Description:    This is the main source code for the coursework
///                   C tic tac toe program
///
///   Author:         Francesco Fico (40404272)     Date: 03/2019
///-----------------------------------------------------------------



//2 dimensional array declared
char board[3][3];

//win condition char return declared
char winCondition(void);

//functions declaration
void createBoard(void);
void displayBoard(void);
void getPlayer1(void);
void getPlayer2(void);
void autoMove(void);
void getAI(void);

//main application
int main(void)
{
    //string to store the win condition declared and set up as blank space
    char win = ' ';

    //printing game title
    printf("\n");
    printf("TIC TAC TOE\n");
    printf("\n");


    //initialise the array
    createBoard();

    //actual game starts
    do
    {
        //print the board on the screen
        displayBoard();
        printf("\n");

        //get the player move
        getPlayer1();
        printf("\n");

        //check if there are win condition
        win = winCondition();

        //if the win char changes from blank someone has won the game
        if (win != ' ')
```

```
            break;

        //re-print the board on the screen
        displayBoard();
        printf("\n");
        printf("AI Moves \n");

        //the computer prints the nought
        autoMove();

        //below the command to play with the euristic AI
        //getAI();

        //below the command to play with a second human player
        //getPlayer2();
        printf("\n");

        //re-check for any winning conditions
        win = winCondition();

      //until win condition does not change stay in the loop
    } while (win == ' ');

    //if the condition string change to an X player 1 wins
    if (win == 'X')
        printf("Player 1 wins!\n");

    //otherwise player 2 wins
    else
        printf("Player 2 wins!\n");

    //print the board again
    displayBoard();

    return 0;
}

//function that initialise the 2 dimensional array
void createBoard(void)
{

    //for loop for the columns
    for (int col = 0; col < 3; col++)

        //for loop for rows
        for (int row = 0; row < 3; row++)

            //fill the 9 positions of the array with blank spaces
            board[col][row] = ' ';
}

//functions that displays the array into a board
void displayBoard(void)
{

    //for loop that prints the board lines
    for (int i = 0; i < 3; i++)
    {

        printf(" %c | %c | %c ", board[i][0], board[i][1], board[i][2]);
```

```
121
122         if (i != 2)
123             printf("\n---|---|---\n");
124     }
125
126     printf("\n");
127 }
128
129 //functions that get the input from the player
130 void getPlayer1(void)
131 {
132     //declares two integers for the coordinates
133     int col, row;
134
135     printf("Player 1 please enter a column (1 to 3) and a row (1 to 3): ");
136
137     //get the user input and stores it
138     scanf("%d%*c%d", &col, &row);
139
140     //subtracts 1 from the user input to match the one stored into the array
141     col--;
142     row--;
143
144     //if the array block is not blank
145     if (board[col][row] != ' ')
146     {
147         //error, the block is not empty
148         printf("ERROR, try again.\n");
149
150         //retry to get another input
151         getPlayer1();
152     }
153
154     //otherwise it stores the value X in the block
155     else
156         board[col][row] = 'X';
157 }
158
159 //functions that get the input from the player 2
160 void getPlayer2(void)
161 {
162     //declares two integers for the coordinates
163     int col, row;
164
165     printf("Player 2 please enter a column (1 to 3) and a row (1 to 3): ");
166
167     //get the user input and stores it
168     scanf("%d%*c%d", &col, &row);
169
170     //subtracts 1 from the user input to match the one stored into the array
171     col--;
172     row--;
173
174     //if the array block is not blank
175     if (board[col][row] != ' ')
176     {
177         //error, the block is not empty
178         printf("ERROR, try again.\n");
179
180         //retry to get another input
```

```
        getPlayer2();
    }

    //otherwise it stores the value X in the block
    else
        board[col][row] = 'O';
}

//return a char that estabilish if there are the winning condition for the game
char winCondition(void)
{

    //for loops that checks the board columns for 3 pieces in line
    for (int col = 0; col < 3; col++)

        if (board[col][0] == board[col][1] &&

            board[col][0] == board[col][2])
            return board[col][0];

    //for loops that checks the board rows for 3 pieces in line
    for (int row = 0; row < 3; row++)

        if (board[0][row] == board[1][row] &&

            board[0][row] == board[2][row])
            return board[0][row];


    //condition to check the board diagonals for 3 pieces in line

    //from the upper left
    if (board[0][0] == board[1][1] &&

        board[1][1] == board[2][2])

        return board[0][0];

    //from the upper right
    if (board[0][2] == board[1][1] &&

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

        return board[0][2];

    //otherwise the win string remains blank
    return ' ';
}

//function the enable the computer to print a nought in the first blank block
available
void autoMove()
{
    //declares two integers for the coordinates
    int col, row;

    //loop the columns
    for (col = 0; col < 3; col++)
    {
        //loop the rows
```

```
        for (row = 0; row < 3; row++)

            //first blank block break the loop
            if (board[col][row] == ' ')
                break;

        //first blank block break loop
        if (board[col][row] == ' ')
            break;
    }

    //if no blank spaces and no win condition
    if (col * row == 9)
    {
        //then the game is draw
        printf("DRAW!\n");

        exit(0);
    }

    //otherwise
    else
        //write in the block
        board[col][row] = 'O';
}

//functions for the more heuristic AI
void getAI(void)
{
    //declares two integers for the coordinates
    int col, row;

    //loop for the columns
    for (col = 0; col < 3; col++)
    {
        //loop for the row
        for (row = 0; row < 3; row++)
        {
            //check for the upper right corner
            if (board[0][0] == ' ' && board[0][0] != 'X')
            {
                //check for the down left corner
                if (board[2][0] == ' ' && board[2][0] != 'X')
                {
                    //check for the upper left corner
                    if (board[0][2] == ' ' && board[0][2] != 'X')
                    {
                        //check for the down right corner
                        if (board[2][2] == ' ' && board[2][2] != 'X')
                        {
                            //if blank write in it
                            board[2][2] = 'O';
                            break;
                        }

                        //otherwise
                        else
                        {
                            //write in it
                            board[0][2] = 'O';
```

```
                            break;
                        }
                    }

                    //otherwise
                    else
                    {
                        //write in it
                        board[2][0] = 'O';
                        break;
                    }
                }

                //otherwise
                else
                {
                    //write in it
                    board[0][0] = 'O';
                    break;
                }
                break;
            }

            //if none of the corner is free write in the first available blank space
            else if (board[col][row] == ' ' && board[col][row] != 'X')
            {
                board[col][row] = 'O';
                break;
            }

            //if no blank spaces and no win condition
            else if (col * row == 9)
            {
                //then the game is draw
                printf("draw\n");

                exit(0);
            }

            break;
        }
        break;
    }
}

```