

DATA STRUCTURES & ALGORITHMS

CSL 221



Bahria University
Discovering Knowledge

Term Project
Tic Tac Toe Game

Saad Ahmed (01-134211-077)
Mohammad Abdullah (01-134211-049)
BS(CS)-3B

TABLE OF CONTENTS

1. INTRODUCTION

- 1.1. Objective
- 1.2. Background
- 1.3. Platform

2. PROJECT DESCRIPTION

- 2.1. Overview
- 2.2. Goals

3. MINIMAX ALGORITHM

- 3.1. Detailed Description
- 3.2. Implementation in Tic-Tac-Toe
- 3.3. Limitation

4. TIC-TAC-TOE CODE IMPLEMENTATION

5. CONCLUSION

Tic Tac Toe

Introduction

1.1. Objective:

The aim is to build a working interactive tic-tac-toe game with added functionalities of playing against another player or playing against the computer where the computer decides the best possible move using an AI algorithm.

1.2. Background:

The tic-tac-toe game is implemented in C++ and hence requires a decent understanding in computer languages in particular C++ and a good grasp on its principles. Minimax algorithm is backtracking AI algorithm which comprises of several data structure and algorithm principles, so it is required to have exposure to these subjects to understand the working of it in more detail.

1.3. Platform:

The platform (IDE) used to develop this program is Visual Studio 2019.

Description of the Project

2.1. Overview:

The tic-tac-toe game is implemented using the computer language C++. There are 2 options given to the user at the start of the program. The option of multiplayer allows the user to play against another user. The other option is single player which allows the user to play against the computer where the computer uses minimax backtracking algorithm to decide the next best move. This is done by mapping all the possible moves and then backtracking. If a certain move benefits the other player, the program learns not to take such moves. The program iterates till either a victor is found or the game ends in a tie. There is also the added functionality of scaling the board from a 3 x 3 to any other size. Increasing the size of the board requires more computation speed. The entire program is implemented in the above way.

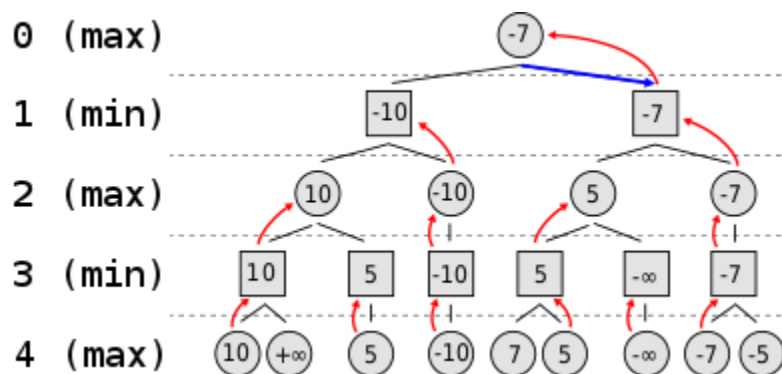
2.2. Goals:

The logic of the game of tic-tac-toe should be separated from the logic of the backtracking algorithm. The initial state of the game is coded. The logic of the tic-tac-toe game is written. The option of multiplayer is first implemented. The logic of the game is then checked by running the program and after debugging the second option of single player is added. The logic of the AI move is implemented using the backtracking algorithm. The program is again debugged if it shows any errors. The functionality of increasing the size of the board is added. After debugging, the program is completed.

Minimax Algorithm

3.1. Description:

Minimax is a decision rule used in various fields like artificial intelligence, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst case scenario. When dealing with minimizing the maximum loss, it is referred to as minimax. When dealing with maximizing the minimum gain, it is referred to as maximin. The intuition behind this algorithm is best explained using a tree data structure. It was originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision-making. In a board game, for every move the computer has to make the minimax algorithm maps all the possible moves in a treelike manner and assigns each node a value based on how much it minimizes the loss. It then takes the next move whose node in that tree falls on a path with least value associated to it. The value for each path is calculated by adding all the values of nodes in that particular path. In minimax algorithm runs by predicting the moves of 2 kinds of players called minimizer and maximizer. Minimizer minimizes the loss whereas the maximizer maximizes the loss which is not beneficial to the AI. The tree that the minimax algorithm generates is similar to the picture shown below.

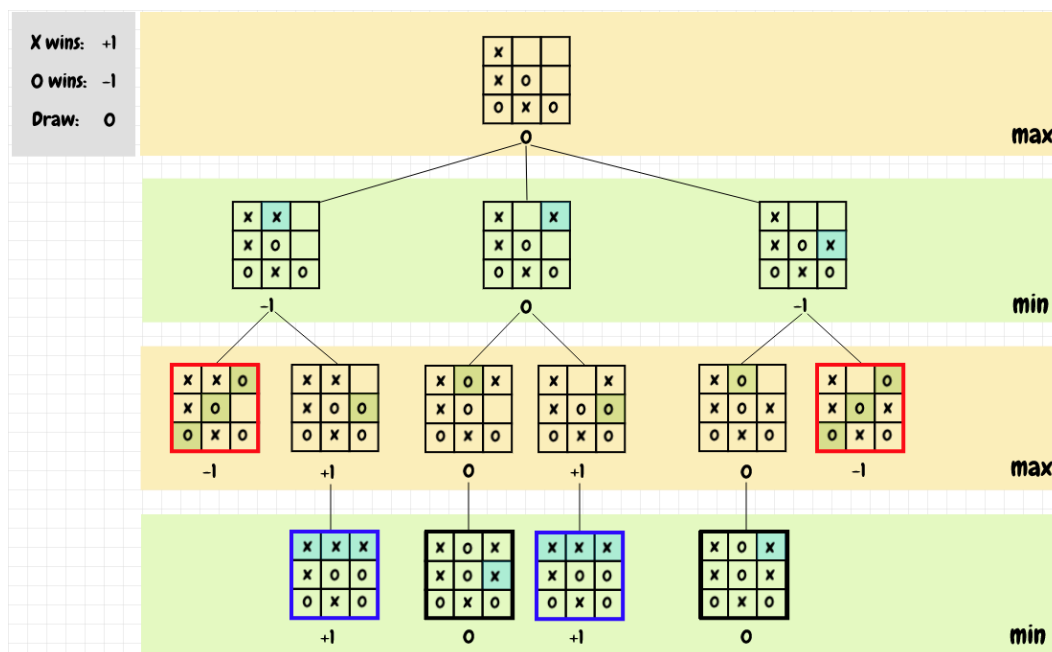


Alternate levels of the tree represents the turns of the maximizer and minimizer. Notice how each node has certain value inside it. That value represents how it minimizes or maximizes the loss for the AI. The algorithm assumes that the

opponent always maximizes the loss for the AI. So the maximizer is the algorithm's prediction of how the human player behaves.

3.2. Implementation in Tic Tac Toe:

The main goal of this game is to get three O's or X's in the same line or diagonal. Tic-Tac-Toe is by far one of the simplest games that this algorithm can run on. For every move that the computer has to decide on, it builds a tree as shown below.



Taking the tree above as an example you can see that the centre node is best decision since its path has the highest value amongst the three. The value of the path was calculated by adding the values of all the nodes below it. In the case of tic-tac toe, there can only be 3 value associated with a node since there can only be 3 distinguishable states. The preferable state is associated with a positive number since it helps the AI player. This preferable state is if the state of the game in which the AI wins. The negative value is associated with a state of the game in which the human player wins. A neutral value or 0 is assigned to every other node or state of the game. The path with the highest value is then taken.

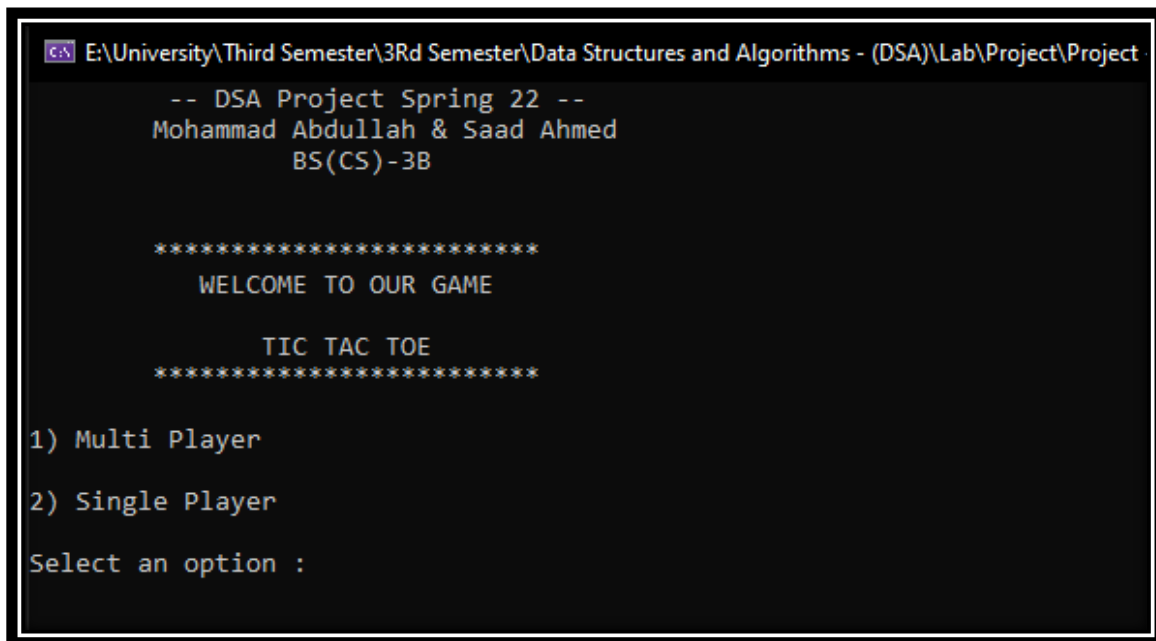
3.3. Limitations:

- ✓ Minimax algorithm assumes that the opponent is always playing the best move it can play. This might not be the case always. The opponent can mislead the algorithm by making a favourable play for the AI.
- ✓ Minimax algorithm works only with purely strategic games. It cannot incorporate any chance component. Games like monopoly, poker and etcetera depend on luck to a certain degree.
- ✓ It can be very computationally expensive. Since it takes into account all the moves that are possible all the way to the end where either the user or the AI wins. The number of moves can range up to 10 to the power of 120 for games like chess. This is considerable much slower to compute.

Implementation

The logic for the game and AI is written separately by defining 2 classes. One called Board which consists of all rules of tic-tac-toe. The other called AI which houses the minimax algorithm. The board stores the state of the game and allows the user and the AI to make moves. It allows the AI and the user to play their turn using the member function called setMove. When the program runs, the user is allowed to choose between single player and multiplayer. The Board class also consist of a variable which stores its size. The program allows us to change that thereby changing the size of the board.

Here are some snips of the program running:



```
E:\University\Third Semester\3Rd Semester\Data Structures and Algorithms - (DSA)\Lab\Project\Project
-- DSA Project Spring 22 --
Mohammad Abdullah & Saad Ahmed
BS(CS)-3B

*****
WELCOME TO OUR GAME

TIC TAC TOE
*****

1) Multi Player
2) Single Player
Select an option :
```



```
E:\University\Third Semester\3Rd Semester\Data Structures and Algorithms - (DSA)\Lab\Project\Project
TIC TAC TOE

|   |   |   |   |   |
|   |   |   |   |   |
|   |   |   |   |   |

It is your turn :
Enter x coordinate : _
```

```
E:\University\Third Semester\3Rd Semester\Data Structures and Algorithms - (DSA)\Lab\Project\Project - TicTacToe\Debug\Project - TicTacToe.exe
TIC TAC TOE

|   o   |   x   |   o   |
|   o   |   x   |   x   |
|   |   |   x   |

X win the game !!!

Press 'C' to continue playing single player, 'M' to return to the menu or 'E' to exit: _
```

Pseudo Code:

```
function minimax(node, depth, maximizingPlayer) is
  if depth == 0 or node is a terminal node then
    return static evaluation of node

  if MaximizingPlayer then
    maxEva= -infinity
    for each child of node do
      eva= minimax(child, depth-1, false)
    maxEva= max(maxEva, eva)
    return maxEva

  else
    minEva= +infinity
    for each child of node do
      eva= minimax(child, depth-1, true)
    minEva= min(minEva, eva)
    return minEva
```

Conclusion:

Tic-Tac-Toe has been successfully implemented using minimax algorithm. Tic-Tac-Toe helped making the Minimax simpler to understand. Minimax has been shown to be intelligent and efficient for simple games like tic-tac-toe. But, It proves to be considerably slow for other complex cases where users can make multiple types of moves like chess. It has also helped build intuition of how backtracking in general works. It has helped bring insight into the vast field of artificial intelligence.

- ✓ Time complexity- As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(bm)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- ✓ Space Complexity- Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.
- ✓ The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from alpha-beta pruning.

Source Code:

```
/*
DSA Project
TIC TAC TOE
Mohammad Abdullah(049) & Saad Ahmed(077)
*/
#include <iostream>
#include <vector>
#include <stdlib.h>
#include <cstdlib>
#include <ctime>
#include <conio.h>

using namespace std;

const int Size = 3;
const int X = 1;
const int O = 2;
const int NA = 0;

class Board //class Board
{
private:
    int t[Size][Size];
public:
    void setMove(int x, int y, int player)
    {
        t[x][y] = player;
    }
    int getVal(int x, int y)
    {
        return t[x][y];
    }
    int checkVictory()
    {
        int c = 0, c1 = 0, c2 = 0, d1 = 0, d2 = 0;
        for (int i = 0; i < Size; i++)
        {
            if (t[i][i] == 1)
                d1++;
            if (t[i][(Size - 1) - i] == 1)
                d2++;
        }
        if (d1 == Size)
            return 1;
        if (d2 == Size)
            return 1;

        d1 = 0; d2 = 0;
        for (int i = 0; i < Size; i++)
        {
            if (t[i][i] == 2)
                d1++;
            if (t[i][(Size - 1) - i] == 2)
                d2++;
        }
    }
}
```

```

    if (d1 == Size)
        return 2;
    if (d2 == Size)
        return 2;

    for (int x = 0; x < Size; x++)
    {
        c1 = 0; c2 = 0;
        for (int y = 0; y < Size; y++)
        {
            if (t[x][y] == 1)
                c1++;
            else if (t[x][y] == 2)
                c2++;
        }
        if (c1 == Size)
            return 1;
        if (c2 == Size)
            return 2;
    }

    for (int y = 0; y < Size; y++)
    {
        c1 = 0;
        c2 = 0;
        for (int x = 0; x < Size; x++)
        {
            if (t[x][y] == 1)
                c1++;
            else if (t[x][y] == 2)
                c2++;
            else
                c++;
        }
        if (c1 == Size)
            return 1;

        if (c2 == Size)
            return 2;
    }
    if (c == 0)
        return 0;
    else
        return -1;
}

void display()
{
    cout << "\t";
    for (int j = 0; j < Size; j++)
        cout << "    ";
    cout << "-\n";
    for (int i = 0; i < Size; i++)
    {
        cout << "\t";
        for (int j = 0; j < Size; j++)
        {
            if (t[i][j] == 0) cout << "| \t \t";

```

```

        else if (t[i][j] == 1) cout << "\\tX\\t"; else
            cout << "\\t0\\t";

    }
    cout << "\\n\\t";
    for (int j = 0; j < Size; j++)
        cout << "    ";
    cout << "-\\n";
}

}

void startBoard()
{
    for (int i = 0; i < Size; i++)
    {
        for (int j = 0; j < Size; j++)
        {
            t[i][j] = 0;
        }
    }
}

bool isFilled(int x, int y)
{
    if (t[x][y] == 0)
        return false;
    else
        return true;
}

void setRandom(int p)
{
    r1:
        srand(time(0));
        int x = rand() % Size;
        int y = rand() % Size;
        if (isFilled(x, y) == false)
            setMove(x, y, p);
        else
            goto r1;
}

};

//Structure AI
struct AImove
{
    int x;
    int y;
    int score;
    AImove() {}
    AImove(int s)
    {
        score = s;
    }
};

//Class AI
class AI
{
private:
    int AIPlayer;
    int humanPlayer;

```

```

public:
    void setAI(int aival, int humanval)
    {
        AIPlayer = aival;
        humanPlayer = humanval;
    }
    AImove getBestMove(Board& b, int player)
    {
        int v = b.checkVictory();

        if (v == AIPlayer)
            return AImove(1);

        else if (v == humanPlayer)
            return AImove(-1);

        else if (v == NA)
            return AImove(0);

        vector <AImove> moves;

        for (int x = 0; x < Size; x++)
        {
            for (int y = 0; y < Size; y++)
            {
                if (b.getVal(x, y) == NA)
                {
                    AImove m;
                    m.x = x;
                    m.y = y;
                    b.setMove(x, y, player);
                    if (player == AIPlayer)
                    {
                        m.score = getBestMove(b, humanPlayer).score;
                    }
                    else
                    {
                        m.score = getBestMove(b, AIPlayer).score;
                    }
                    moves.push_back(m);
                    b.setMove(x, y, NA);
                }
            }
        }
        int bm = 0;
        int bs;
        int i{};
        if (player == AIPlayer)
        {
            int bs = -99999;
            for (int i = 0; i < moves.size(); i++)
            {
                if (moves[i].score > bs)
                {
                    bs = moves[i].score;
                }
            }
        }
    }

```

```

        else
            bm = i;

        bs = moves[i].score;
        bs = 99999;
        for (int i = 0; i < moves.size(); i++)
        {
            if (moves[i].score < bs)
            {
                bm = i; bs = moves[i].score;
            }
        }
        return moves[bm];
    }
    void perform(Board& b)
    {
        AImove bm = getBestMove(b, AIPlayer);
        b.setMove(bm.x, bm.y, AIPlayer);
    }
};

//start main
int main()
{
    Board b;
    int o, x, y, c, Player;
    char p, s, q;
mainMenu:
    system("cls");
    cout << "\t -- DSA Project Spring 22 -- \n";
    cout << "\tMohammad Abdullah & Saad Ahmed\n";
    cout << "\t\t BS(CS)-3B\n";
    cout << "\n\n\t*****\n";
    cout << "\t\t WELCOME TO OUR GAME" << endl;
    cout << "\n\t\t TIC TAC TOE" << endl;
    cout << "\t*****\n";
11:
    cout << "\n1) Multi Player \n" << endl;
    cout << "2) Single Player \n" << endl;
    cout << "Select an option : ";
    cin >> o;
    if (o == 1) {
        multiPlayer: b.startBoard();
        cout << "\n";
        while (b.checkVictory() == -1)
        {
            while (Player == X && b.checkVictory() == -1)
            {
                system("cls");
                cout << "\n\t\t\t TIC TAC TOE" << endl;
                cout << "\n\n";
                b.display();
                cout << "\n\n";

12:
                cout << "\t" << p << " 's turn : " << endl;
                cout << "\tEnter x coordinate : "; cin >> x;
                cout << "\tEnter y coordinate : "; cin >> y;
            }
        }
    }
}

```

```

        cout << "\n";
        if (x <= Size && y <= Size && x > 0 && y > 0 && b.isFilled(x -
1, y - 1) == false)
        {
            b.setMove(x - 1, y - 1, Player);
            Player = 0;
            p = 'O';
        }
        else
        {
            cout << "\tInvalid location \n\n";
            goto l2;
        }
    }
    while (Player == 0 && b.checkVictory() == -1)
    {
        system("cls");
        cout << "\n\t\t\t TIC TAC TOE" << endl;
        cout << "\n\n";
        b.display();
        cout << "\n\n";
l3:
        cout << "\t" << p << " 's turn : " << endl;
        cout << "\tEnter x coordinate : ";
        cin >> x;
        cout << "\tEnter y coordinate : ";
        cin >> y;
        cout << "\n";
        if (x <= Size && y <= Size && x > 0 && y > 0 && b.isFilled(x -
1, y - 1) == false)
        {
            b.setMove(x - 1, y - 1, Player);
            Player = X;
            p = 'X';
        }
        else
        {
            cout << "\tInvalid location \n\n";
            goto l3;
        }
    }
    system("cls");
    cout << "\n\t\t\t TIC TAC TOE" << endl;
    cout << "\n\n";

    b.display(); cout << "\n\n"; if (b.checkVictory() == 1)
    {
        cout << "\tX win the game !!!\n";
    }
    else if (b.checkVictory() == 2)
    {
        cout << "\tO win the game !!!\n";
    }
    else
    {
        cout << "\tIts a Tie !!!\n";
    }
}

```



```

c1:
    cout << "\n\tPress 'C' to continue playing multi player, 'M' to return to
the menu or 'E' to exit:";
    cin >> q;
    if (q == 'M' || q == 'm')
        goto mainMenu;
    else if (q == 'C' || q == 'c')
        goto multiPlayer;
    else if (q == 'E' || q == 'e')
        system(0);
    else
    {
        cout << "\nEnter a valid option ! ";
        goto c1;
    }
}
else if (o == 2)
{
    AI a;
    int hp, aip;
    char xoo;
    goto s1;
singlePlayer:
    system("cls");
s1:
    c = 0;
    b.startBoard();
14:
    cout << "\n\tSelect 'X' or 'O' : ";
    cin >> xoo;
    if (xoo == 'X' || xoo == 'x')
    {
        a.setAI(2, 1); hp = 1;

        aip = 2;
    }
    else if (xoo == 'O' || xoo == 'o')
    {
        a.setAI(1, 2);
        hp = 2;
        aip = 1;
    }
    else
    {
        cout << "\n\tEnter valid option !\n";
        goto 14;
    }
15:
    cout << "\n\tDo you want to start the game ? ( Y / N ) : ";
    cin >> s;
    if (s == 'y' || s == 'Y')
        Player = hp;
    else if (s == 'n' || s == 'N')
        Player = aip;
    else
    {
        cout << "\n\tEnter valid option !\n";
        goto 15;
    }

```

```

    }

    while (b.checkVictory() == -1)
    {
        while (Player == hp && b.checkVictory() == -1)
        {
            system("cls");
            cout << "\n\t\t\t TIC TAC TOE" << endl;
            cout << "\n\n";
            b.display();
            cout << "\n\n"; l6:
            cout << "\tIt is your turn : " << endl;
            cout << "\tEnter x coordinate : ";
            cin >> x;
            cout << "\tEnter y coordinate : ";
            cin >> y;
            cout << "\n";
            if (x <= Size && y <= Size && x > 0 && y > 0 && b.isFilled(x -
1, y - 1) == false)
            {
                b.setMove(x - 1, y - 1, Player);
                Player = aip;
                system("cls");

                cout << "\n\t\t\t TIC TAC TOE" << endl;
                cout << "\n\n";
                b.display();
                cout << "\n\n";
                cout << "\tThe computer is thinking      " << endl;
            }
            else
            {
                cout << "\tInvalid location \n\n";
                goto l6;
            }
        }
        while (Player == aip && b.checkVictory() == -1)
        {
            if (c < 1 && (s == 'N' || s == 'n'))
            {
                b.setRandom(Player); c++;
            }
            else
                a.perform(b); Player = hp;
        }
    }
    system("cls");
    cout << "\n\t\t\t TIC TAC TOE" << endl; cout << "\n\n";
    b.display(); cout << "\n\n"; if (b.checkVictory() == 1)
    {
        cout << "\tX win the game !!!\n";
    }
    else if (b.checkVictory() == 2)
    {
        cout << "\tO win the game !!!\n";
    }
    else
    {

```

```

        cout << "\tIts a Tie !!! \n";
    }
c2:    cout << "\n\tPress 'C' to continue playing single player, 'M' to return to
the menu or 'E' to exit:";
    cin >> q;
    if (q == 'M' || q == 'm')
        goto mainMenu;
    else if (q == 'C' || q == 'c')
        goto singlePlayer;
    else if (q == 'E' || q == 'e')
        system(0);
    else
    {
    }
}
else
{
    cout << "\nEnter a valid option ! ";
    goto c2;

    cout << "\n\tInvalid option! \n";
    goto l1;
}
_getch();
return 0;
} //end main

```

Thank You...