East West University

Department of Computer Science and Engineering

Project Report

Course: CSE366

Section: 01

Semester: Fall 2022

Submitted By:

Name	ID
Farhan Asfar	2020-1-60-154
MD Azman Ahmed	2020-1-60-217
Anik Lal Dey	2020-1-60-228

Submitted to

Jesan Ahammed Ovi

East West University

Date of Submission:

12th January 2023

1. Problem Statement

Build a Tic-Tac-Toe game where the game will be played between a player and a computer. Considering the player as an optimal opponent, the computer must play optimally too. In every move, the computer will play the best possible move and decrease the chance of winning for the player.

2. System Requirements

Memory: 512 MB

CPU: Intel Pentium 4, 2.00 GHz

Disk Space: 5 MB

OS: Windows 7 or above

3. System Design

The game is initially designed to play between a player and the computer.

We are considering that the player will be playing optimally, so in reply the computer should play the best move too.

The game is implemented based on the ‘Adversarial Search’ algorithm.

Initial State

Initially there will be a 3x3 board to play the game. Anyone of the computer or the player will start the game. And then the computer will evaluate the current state of the board.

Successor Function

After the initial state, the computer will start to evaluate the possible moves on the board and find the best move. Here, we have a ‘bestMove’ function to find that. The computer will place a move on the next available position and will calculate the best score for that position. For each possible move the computer will use an utility function to find the best score.

To calculate the best score for the current position, we will use the minimax algorithm.

Utility Function

Here, the ‘minimax’ function will be used as the utility function to determine the best value considering the state of the board.

The function will find the best value for both the player and the computer. After considering each possible position, the function will return the best score for which the computer will make its next move.

The purpose of this function is to maximize the score for the computer and minimize the chance of the player to win.

Terminal Test

To evaluate the current state of the board whether the game is over or not we will use a terminal function. Here, we have used a terminal function called ‘gameOver’ to check that.

The ‘gameOver’ function will check the board row wise, column wise and also diagonally to determine the winner. If the board is not crossed row wise, column wise or diagonally and if the move index has reached the maximum value then the game is drawn.

4. Implementation

At first, in the **main** function we are giving the user the choice whether he wants to move first or let the computer move first.

In **playTicTacToe** function,

1. We are declaring and initializing the board with blank space,
2. Then we are starting a while which will run until either the game is over or it is a draw,
3. Here we are checking whose move it is,

   If computers move: Computer is finding the best move using the **bestMove** function.

   Else if its Humans move, at first the program will show possible moves available to the user. Then user select his move, the program will check the whether the move is valid or not, if valid the board will be updated

4. In the **bestMove** function:

```
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*SIDE+y;
}
```

This function will only be used when its computers move. The nested loop will traverse the whole grid and if there is any empty space the **minimax** function will check if it is the best move or not. If the best move is found then the cell index would be returned to the **playTicTacToe** function.

5. In the **minimax** function,

```
///MINIMAX ALGORITHM
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 < SIDE*SIDE)
    {
      if(isAI == true) //true mani computer er move, ekhane bestscore maximum korte hobe
      {
        bestScore = -1e5;
        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] = ' ';
              bestScore = max(bestScore, score);

            }
          }
        }
        return bestScore;
      }
```

```
    else  //false mani human er move, ekhane bestscore minimum korte hobe
    {
      bestScore = 1e5;
      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] = ' ';
            bestScore = min(bestScore,score);
          }
        }
      }
      return bestScore;
    }
  }
  else return 0;

}
}
```

It takes the board, number of moves played as depth and an initial boolean value **false.** It is a recursive function. The base case is that it checks if a winner has been found. Otherwise it checks if the depth is valid or not. If that boolean variable is false then it is computers move, vice versa.

If computers move then the machine will try to maximize the value of best score to prevent the user from winning. When it is Humans move the minimax algorithm will for which move the value best score will be minimum, since the user will always try to win using the optimal move.

6. The function gameOver checks for winning cases by using 3 other functions named rowCrossed, columnCrossed and diagonalCrossed.

5. Testing Results

```
You can insert in the following positions :
2 8 9
Enter the position =
8

HUMAN has put a X in cell 8

O |  |X
---------------
X |X |O
---------------
O |X |

COMPUTER has put a O in cell 2
O |O |X
---------------
X |X |O
---------------
O |X |

You can insert in the following positions :
9
Enter the position =
9

HUMAN has put a X in cell 9

O |O |X
---------------
X |X |O
---------------
O |X |X

It's a draw
```

```
                        Tic-Tac-Toe

Do you want to start first?(y/n) :
y

Choose a cell numbered from 1 to 9 as below and play
1   |2   |3
---------------
4   |5   |6
---------------
7   |8   |9

You can insert in the following positions :
1 2 3 4 5 6 7 8 9
Enter the position =
```

```
You can insert in the following positions :
1 2 3 4 5 6 7 8 9
Enter the position =
5

HUMAN has put a X in cell 5

  |  |
---------------
  |X |
---------------
  |  |

COMPUTER has put a O in cell 1
O |  |
---------------
  |X |
---------------
  |  |

You can insert in the following positions :
2 3 4 6 7 8 9
Enter the position =
```

```
HUMAN has put a X in cell 2

X |X |
---------------
  |O |
---------------
  |  |

COMPUTER has put a O in cell 3
X |X |O
---------------
  |O |
---------------
  |  |

You can insert in the following positions :
4 6 7 8 9
Enter the position =
6

HUMAN has put a X in cell 6

X |X |O
---------------
  |O |X
---------------
  |  |

COMPUTER has put a O in cell 7
X |X |O
---------------
  |O |X
---------------
O |  |

COMPUTER has won
```

6. Limitations & Future Scope:

It is a good and easy game to play and playing this we don’t need any human opponent we can easily play this game with the computer that's why in the future we have a plan for updating this game. In the future, our plan is to update this game with more features. Features will be implementing a nice graphical visualization for playing the game ,human vs human ,computer vs computer playing system and adding one more game to it like chess in the same way and also using this same minimax algorithm.

Limitation Our project has some limitations, that is our project playing system is only human against computer only and when we increase the board size then the time complexity increases because we don’t use here alpha beta pruning.