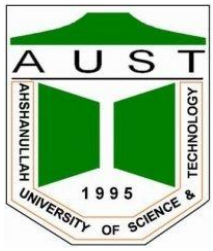
Ahsanullah University of Science & Technology
Department of Computer Science & Engineering



AI LAB POJECT 4

Artificial Intelligence Lab
CSE-4108

Implement a Minimax Algorithm to simulate Tic Tac Toe game(3x3)

Submitted By
Name: Ashiqur Rahman
ID: 15-02-04-057
Section: A2

Question : Implement a Minimax Algorithm to simulate Tic Tac Toe game(3x3)

Answer :

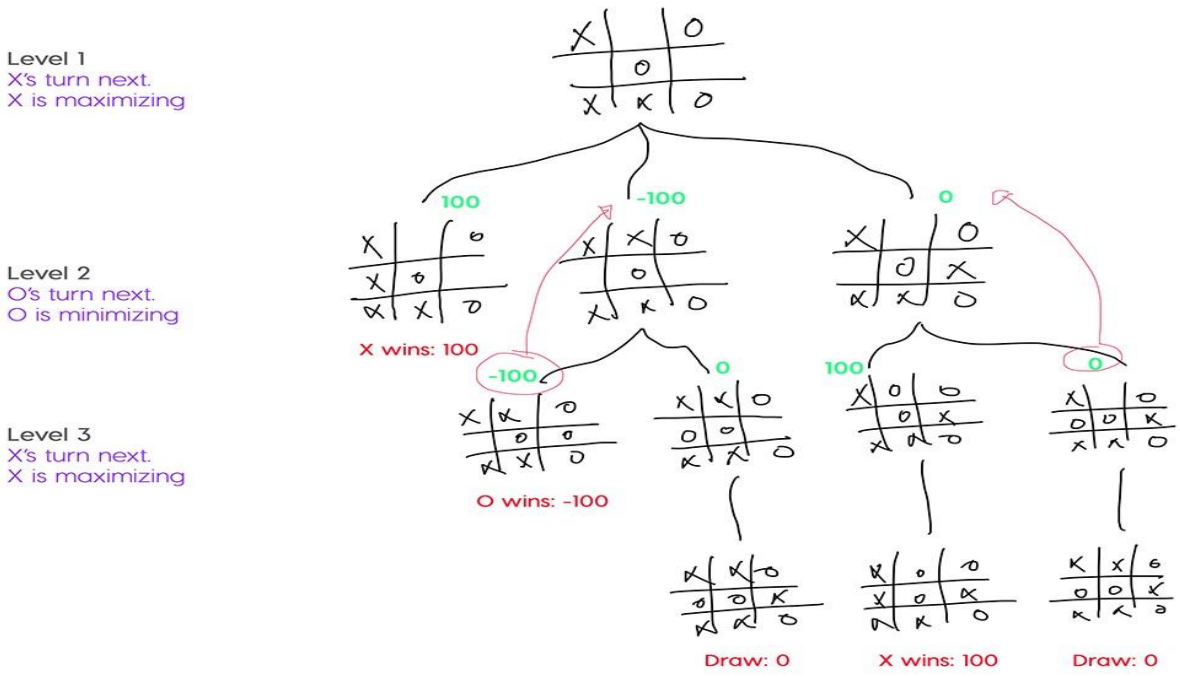
To apply minimax algorithm in two-player games, we are going to assume that X is a *maximizing player* and O is a *minimizing player*. The maximizing player will try to maximize its score or in other words choose the move with the highest value. The minimizing player will try to minimize the value for the maximizing player, thus choosing the move with the minimum value.

In order to calculate the values mentioned above, we need to decide on some assumptions. We call these values heuristic values. In tic-tac-toe we have 3 possibilities:

- The state of the board is a draw: We will give this board a value of 0;
- X wins in a board state: We will give this board a value of 100;
- O wins in a board state: We will give this board a value of -100;

Minimax Example with Game Tree :

To illustrate the minimax algorithm more, let's take a look at a visual example. In the diagram below, consider a situation where it's X's turn given the current state. X have three possible moves:



- Level 1: X has three possible moves and tries to find the maximum node.
- Level 2: The first move leads to direct win for X thus 100 points is given.
- Level 2: The second and third moves will lead to two more possible moves where it's O's turn.
- Level 3: O is trying to minimize the score so it chooses the nodes with the minimum value.
- Level 3: The first move for O will lead to a win and the second to a draw, thus we assume that O is going to choose the first move and the parent node will have a value of -100. Same for the third and fourth moves.
- Back to Level 1, X now has to choose between 100, -100 and 0. Since X is the maximizer, it will definitely choose 100 which will lead to a win.

As you can notice, we recursively propagate the possibilities tree calculating the score for each terminal state and then going back to decide which move we will take.

CODE:

```
import time

import sys

from sys import stdout




class TileState:

    EMPTY = 0

    CIRCLE = 1

    CROSS = 2




class Board:

    SIZE = 3




class Mode:

    CIRCLE = 1
```

```
        CROSS = 2



class State:

        FULL = 0

        CROSS_WIN = 1

        CIRCLE_WIN = 2

        NOTFULL = 3



def print_board(board):

        print ("")

        for i in range(Board.SIZE**2):

                if board[i] == TileState.EMPTY:

                        stdout.write("   ")

                elif board[i] == TileState.CIRCLE:

                        stdout.write(" 0 ")

                elif board[i] == TileState.CROSS:
```

```
                stdout.write(" X ")
            if (i+1) % Board.SIZE == 0:
                if (i+1) != Board.SIZE**2:
                    print ("")
                    for r in range(Board.SIZE):
                        stdout.write("--- ")
                    print ("")
            else:
                stdout.write("|")
        print ("")


def checkState(board):

    isFull = True
    crossWin = [True]*(Board.SIZE*2+2)
    circleWin = [True]*(Board.SIZE*2+2)
```

```
for i in range(Board.SIZE):

        if board[i*Board.SIZE+i] != TileState.CROSS:
            crossWin[Board.SIZE*2] = False

        if board[i*Board.SIZE+i] != TileState.CIRCLE:
            circleWin[Board.SIZE*2] = False
        if board[i*Board.SIZE+Board.SIZE-i-1] != TileState.CROSS:
            crossWin[Board.SIZE*2+1] = False
        if board[i*Board.SIZE+Board.SIZE-i-1] != TileState.CIRCLE:
            circleWin[Board.SIZE*2+1] = False

        for j in range(Board.SIZE):

            if board[i*Board.SIZE+j] == TileState.EMPTY:
                isFull = False

            if board[i*Board.SIZE+j] != TileState.CROSS:
```

```
                    crossWin[i] = crossWin[Board.SIZE+j] = False


                if board[i*Board.SIZE+j] != TileState.CIRCLE:

                    circleWin[i] = circleWin[Board.SIZE+j] = False


for c in crossWin:
    if c == True:
        return State.CROSS_WIN
for c in circleWin:
    if c == True:
        return State.CIRCLE_WIN
if isFull:
    return State.FULL
else:
    return State.NOTFULL


if isFull == True:
    return State.FULL
```

```
        else:
                return State.NOTFULL



def move(board):
        state = checkState(board)


        if state != State.NOTFULL:
                return state
        maxp = -1
        nextstep = 0
        beta=100
        for i in range(Board.SIZE**2):
                alpha = -1
                if board[i] == TileState.EMPTY:
                        newboard = board[:]
                        newboard[i] = Mode.CROSS
```

```
                    p = minMaxSearch(newboard,Mode.CIRCLE, alpha, beta)

                    if p > maxp:
                        maxp = p
                        nextstep = i

        board[nextstep] = Mode.CROSS

        return checkState(board)


def minMaxSearch(board, mode, alpha, beta):
        state = checkState(board)

        if state == State.CROSS_WIN:
            return 1
        elif state == State.CIRCLE_WIN:
```

```
        return 0


elif state == State.FULL:
        return 0.1



if mode == Mode.CIRCLE:
        newmode = Mode.CROSS
else:
        newmode = Mode.CIRCLE
maxp = -1
minp = 100
for i in range(Board.SIZE**2):
        tile = board[i]
        if tile == TileState.EMPTY:
                newboard = board[:]
                newboard[i] = mode
                p = minMaxSearch(newboard, newmode, alpha, beta)
```

```
                if mode == Mode.CROSS:
                    if p >= maxp:
                        maxp = p
                        if maxp > alpha:
                            alpha = maxp

                elif mode == Mode.CIRCLE:
                    if p <= minp:
                        minp = p

                        beta = minp

if mode == Mode.CROSS:
    return maxp
```

```
        elif mode == Mode.CIRCLE:
                return minp
        else:
                print( " impossible " )



def main():


        board = [TileState.EMPTY]*Board.SIZE**2


        print_board(board)
        while True:
                try:


                        c = int(input("Enter your move(1-9): "))
                        if c <= 0 or c > Board.SIZE**2:
                                raise ValueError
                except ValueError:
```

```
                print( "Please make a move by entering a number between
%d and %d" %(1,Board.SIZE**2))

                continue



        c = c - 1

        if c >= 6:

            c = c - 6

        elif c <= 2:

            c = c + 6

        if board[c] == TileState.EMPTY:

            board[c] = TileState.CIRCLE

        else:

            print( "The move is illegal")

            continue



        print_board(board)

        state = move(board)

        print_board(board)
```

```
            if state == State.CIRCLE_WIN:
                print ("You WIN !!!!!!1!")
                break
            elif state == State.CROSS_WIN:
                print ("You LOSE !!")
                print()
                break
            elif state == State.FULL:
                print ("Draw!!")
                print()
                print()
                break


main()
```