Anjali Agrawal

(RA1811030010096)

Lab Exercise #6

Implementation of Minimax Algorithm for an application

Title:

Implementation of minimax algorithm for an application

Problem Description:

We have to implement the minimax algorithm in an application where e are given a tic-tac-toe board and we need to predict which is the best move for us using the minimax algorithm.

In the game problem A player can either take X or O. Then chance by change one after the other, each take their turn on the player board and mark their symbol on the playboard. Once placed the next player takes the turn and embark its symbol without overlapping any existing symbols on the board.

The player who is able to create a sequence of three continuous blocks of their symbol wins the game. The sequence can be either horizontal, vertical or diagonal. The computer must be able to detect and predict the best possible step based on the choice it takes, it needs to place the token assigned on the board and hence clear the condition on the board.

Solution:

We solve the problem using minimax algorithm

Finding the Best Move :

We shall be introducing a new function called findBestMove(). This function evaluates all the available moves using minimax() and then returns the best move the maximizer can make. The pseudocode is as follows :

```
function findBestMove(board):
    bestMove = NULL
    for each move in board :
        if current move is better than bestMove
            bestMove = current move
    return bestMove
```

Minimax :
To check whether or not the current move is better than the best move we take the help of minimax() function which will consider all the possible ways the game can go and returns the best value for that move, assuming the opponent also plays optimally
The code for the maximizer and minimizer in the minimax() function is similar to findBestMove() , the only difference is, instead of returning a move, it will return a value. Here is the pseudocode :

```
function minimax(board, depth, isMaximizingPlayer):

    if current board state is a terminal state :
        return value of the board

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each move in board :
            value = minimax(board, depth+1, false)
            bestVal = max( bestVal, value)
        return bestVal

    else :
        bestVal = +INFINITY
        for each move in board :
            value = minimax(board, depth+1, true)
            bestVal = min( bestVal, value)
        return bestVal
```

Checking for GameOver state :
To check whether the game is over and to make sure there are no moves left we use isMovesLeft() function. It is a simple straightforward function which checks whether a move is available or not and returns true or false respectively.
Pseudocode is as follows :

```
function isMovesLeft(board):
    for each cell in board:
        if current cell is empty:
            return true
    return false
```

Python Code:

```
player, opponent = 'x', 'o'
def isMovesLeft(board) :

    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                return True
    return False

def evaluate(b) :
    for row in range(3) :
        if (b[row][0] == b[row][1] and b[row][1] == b[row][2]) :
            if (b[row][0] == player) :
                return 10
            elif (b[row][0] == opponent) :
                return -10

    for col in range(3) :

        if (b[0][col] == b[1][col] and b[1][col] == b[2][col]) :

            if (b[0][col] == player) :
                return 10
            elif (b[0][col] == opponent) :
                return -10

    if (b[0][0] == b[1][1] and b[1][1] == b[2][2]) :

        if (b[0][0] == player) :
            return 10
        elif (b[0][0] == opponent) :
            return -10

    if (b[0][2] == b[1][1] and b[1][1] == b[2][0]) :

        if (b[0][2] == player) :
            return 10
        elif (b[0][2] == opponent) :
            return -10

    return 0


def minimax(board, depth, isMax) :
    score = evaluate(board)
    if (score == 10) :
        return score
    if (score == -10) :
        return score
    if (isMovesLeft(board) == False) :
        return 0
    if (isMax) :
        best = -1000
        for i in range(3) :
            for j in range(3) :
                if (board[i][j]=='_') :
                    board[i][j] = player
                    best = max( best, minimax(board,
                                        depth + 1,
                                        not isMax) )
                    board[i][j] = '_'
        return best
    else :
        best = 1000
        for i in range(3) :
            for j in range(3) :
                if (board[i][j] == '_') :
                    board[i][j] = opponent
                    best = min(best, minimax(board, depth + 1, not isMax))

                    board[i][j] = '_'
        return best


def findBestMove(board) :
    bestVal = -1000
    bestMove = (-1, -1)
    for i in range(3) :
        for j in range(3) :
            if (board[i][j] == '_') :
                board[i][j] = player
                moveVal = minimax(board, 0, False)
                board[i][j] = '_'
                if (moveVal > bestVal) :
                    bestMove = (i, j)
                    bestVal = moveVal

    print()
    return bestMove

board = [
    [ 'x', 'o', 'x' ],
    [ 'o', 'x', 'o' ],
    [ '_', '_', '_' ]
]

bestMove = findBestMove(board)

print("The Optimal Move is :")
print("ROW:", bestMove[0], " COL:", bestMove[1])
```

Input and output:

Sample input board:

```
board = [
    [ 'x', 'o', 'x' ],
    [ 'o', 'x', 'o' ],
    [ '_', '_', '_' ]
]
```

Sample output:

```
The Optimal Move is :
ROW: 2  COL: 0
```

Result:

Hence the application of minimax algorithm of finding the optimal move in a tic tac toe puzzle is implemented.