Batch 1

PA 17- Ketaki Patil

MIT-WPU | Dr Vishwanath Karad MIT WORLD PEACE UNIVERSITY | PUNE

AI : Lab Assignment 2

Title : Implementation of min-max algorithm for Tic-Tac-Toe game.

Aim : Solve Tic Tac Toe using minmax algorithm.

Objective : To study and implement minmax algorithm for tic tac toe.

Theory :

1) **Adversarial search :**

Adversarial search is search when there is an opponent changing the state of the problem every step in a direction you do not want. Eg. chess, business, trading

You change state, but then you don't control next state opponent will change next state in a way :

a) unpredictable

b) hostile to you

You only get to change every alternate state.

2) **Tic Tac Toe solving steps :**

Consider two opponents, 1st represented by 'x' and the other by 'o', where we aim on maximizing the chance of 'x' winning. Rules are as follows:

a) If 'x' wins, take it.

b) If opponent wins, block it

c) If possible create a fork (2 winning ways)

d) Do not let opponent block 'X' winning move.
e) If neither 'X' or 'O' wins call it a tie.

3) Data structures and other algo details about Minmax algorithm excluding algorithm

Minmax is a backtracking algorithm that is used in decision making and game theory to find optimal move for a player. A binary tree is used for this algorithm it has 2 players maximizer, who tries to get highest score possible and minimizer who tries to get lowest score possible. It is widely used in 2 player turn based games such as tictactoe, chess etc. Performs depth first search algorithm.

Input : Initial State
Output : Solution goal state with optimal path.
Algorithm : Minmax

PAQ :

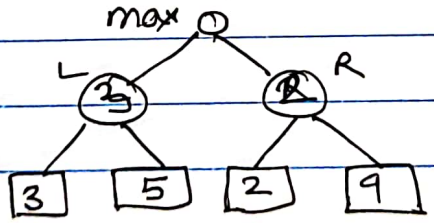
1) Compare informed search and adversarial search.

Informed search	Adversarial search
(a) uses knowledge for searching process.	(a) Doesn't use knowledge for searching process.
(b) Finds solution quickly.	(b) Finds solution slowly.
(c) Less time	(c) Moderate time.
(d) Less lengthy while implementation.	(d) More lengthy while implementation.

2) Explain minmax algorithm with example.

→ Every board state has value associated with it. In a given state if maximizer has upper hand then, score of board will tend to be some positive value. If minimizer has upper hand in that board state then it will tend to be some negative value. Values of board are calculated by some heuristics which are unique for every type of game.

Eg. Consider game with 4 final states and maximizing player starting first. The game tries all possible moves since it is backtracking algorithm. Maximizer goes left – It is minimizer turn now, that has choice between 3 and 5. Being minimizer it will choose least among both that is 3. Maximizer goes right – It is minimizers turn. It has now a choice between 2 and 9. It will choose 2. Maximizer will choose will choose largest value 3. Hence optimal move for maximizer is to go left.



3) Explain alpha-beta pruning.

→ It is a optimization technique for minmax algorithm. It reduces computation time and allows us to search much faster and even go into deeper levels in game tree. It cuts off branches in game tree & which need not be searched because there already exists a better move

available. It passes 2 extra parameters in minmax function.

Alpha - Best value that maximizer currently can guarantee at that level or above.

Beta - Best value that minimizer currently can guarantee at that level or above.

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

CODE :

```
'''
ASSIGNMENT 2 : TIC TAC TOE IMPLEMENTATION WITH MIN-MAX ALGORITHM
PA 17 KETAKI PATIL
BATCH A1
'''

import time

class Game:
    def __init__(self):
        self.initialize_game()

    def initialize_game(self):
        self.current_state = [['.','.','.'],
                              ['.','.','.'],
                              ['.','.','.']]

        # Player X always plays first
        self.player_turn = 'X'

    def draw_board(self):
        for i in range(0, 3):
            for j in range(0, 3):
                print('{}|'.format(self.current_state[i][j]), end=" ")
            print()
        print()


    # Determines if the made move is a legal move
    def is_valid(self, px, py):
        if px < 0 or px > 2 or py < 0 or py > 2:
            return False
        elif self.current_state[px][py] != '.':
            return False
        else:
            return True


    # Checks if the game has ended and returns the winner in each case
    def is_end(self):
        # Vertical win
        for i in range(0, 3):
            if (self.current_state[0][i] != '.' and
                self.current_state[0][i] == self.current_state[1][i] and
                self.current_state[1][i] == self.current_state[2][i]):
                return self.current_state[0][i]
```

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

```
        # Horizontal win
        for i in range(0, 3):
            if (self.current_state[i] == ['X', 'X', 'X']):
                return 'X'
            elif (self.current_state[i] == ['O', 'O', 'O']):
                return 'O'

        # Main diagonal win
        if (self.current_state[0][0] != '.' and
            self.current_state[0][0] == self.current_state[1][1] and
            self.current_state[0][0] == self.current_state[2][2]):
            return self.current_state[0][0]

        # Second diagonal win
        if (self.current_state[0][2] != '.' and
            self.current_state[0][2] == self.current_state[1][1] and
            self.current_state[0][2] == self.current_state[2][0]):
            return self.current_state[0][2]

        # Is whole board full?
        for i in range(0, 3):
            for j in range(0, 3):
                # There's an empty field, we continue the game
                if (self.current_state[i][j] == '.'):
                    return None

        # It's a tie!
        return '.'



    # Player 'O' is max, in this case AI
    def max(self):

        # Possible values for maxv are:
        # -1 - loss; 0  - a tie; 1  - win
        # We're initially setting it to -2 as worse than the worst case:
        maxv = -2
        px = None
        py = None
        result = self.is_end()
        # If the game came to an end, the function needs to return
        # the evaluation function of the end. That can be:
        # -1 - loss; 0  - a tie;  1  - win
        if result == 'X':
            return (-1, 0, 0)
        elif result == 'O':
            return (1, 0, 0)
        elif result == '.':
            return (0, 0, 0)
```

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

```
    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                # On the empty field player 'O' makes a move and calls Min
                # That's one branch of the game tree.
                self.current_state[i][j] = 'O'
                (m, min_i, min_j) = self.min()
                # Fixing the maxv value if needed
                if m > maxv:
                    maxv = m
                    px = i
                    py = j
                # Setting back the field to empty
                self.current_state[i][j] = '.'
    return (maxv, px, py)


# Player 'X' is min, in this case human
def min(self):
    # Possible values for minv are:
    # -1 - win; 0  - a tie; 1  - loss
    # We're initially setting it to 2 as worse than the worst case:
    minv = 2
    qx = None
    qy = None
    result = self.is_end()

    if result == 'X':
        return (-1, 0, 0)
    elif result == 'O':
        return (1, 0, 0)
    elif result == '.':
        return (0, 0, 0)

    for i in range(0, 3):
        for j in range(0, 3):
            if self.current_state[i][j] == '.':
                self.current_state[i][j] = 'X'
                (m, max_i, max_j) = self.max()
                if m < minv:
                    minv = m
                    qx = i
                    qy = j
                self.current_state[i][j] = '.'

    return (minv, qx, qy)


def play(self):
```

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

```
        while True:
            print('Current Board position : ')
            self.draw_board()
            self.result = self.is_end()

            # Printing the appropriate message if the game has ended
            if self.result != None:
                if self.result == 'X':
                    print('The winner is X!')
                elif self.result == 'O':
                    print('The winner is O!')
                elif self.result == '.':
                    print("It's a tie!")

                self.initialize_game()
                return

            # If it's player's turn
            if self.player_turn == 'X':

                while True:

                    start = time.time()
                    (m, qx, qy) = self.min()
                    end = time.time()
                    print('Evaluation time: {}s'.format(round(end - start, 7)))
                    print('Recommended move: X = {}, Y = {}'.format(qx, qy))
                    n=int(input('Insert your move: '))
                    if(n==1):
                      px=py=0
                    if(n==2):
                      px=0
                      py=1
                    if(n==3):
                      px=0
                      py=2
                    if(n==4):
                      px=1
                      py=0
                    if(n==5):
                      px=py=1
                    if(n==6):
                      px=1
                      py=2
                    if(n==7):
                      px=2
                      py=0
                    if(n==8):
                      px=2
                      py=1
```

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

```
                    if(n==9):
                      px=py=2

                    (qx, qy) = (px, py)

                    if self.is_valid(px, py):
                        self.current_state[px][py] = 'X'
                        self.player_turn = 'O'
                        break
                    else:
                        print('The move is not valid! Try again.')

            # If it's AI's turn
            else:
                (m, px, py) = self.max()
                self.current_state[px][py] = 'O'
                self.player_turn = 'X'


def main():
    g = Game()
    print('Board positions are like this: ')
    for i in range(3):
        print(
            "| " + str(i * 3 + 1) +
            " | " + str(i * 3 + 2) +
            " | " + str(i * 3 + 3) + " |"
        )
    g.play()

if __name__ == "__main__":
    main()
```

AI ASSIGNMENT 2 : TICTACTOE - MINMAX

PA 17 KETAKI PATIL

OUTPUT :

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\SEM_9\AI>  d:; cd 'd:\SEM_9\AI'; & 'C:\Users\ketak\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.7_qbz
scode\extensions\ms-python.python-2021.5.842923320\pythonFiles\lib\python\debugpy\launcher' '50140' '--' 'd:\SEM_9\AI\tictactoe.py'
Board positions are like this:
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
Current Board position :
.| .| .|
.| .| .|
.| .| .|

Evaluation time: 4.9641428s
Recommended move: X = 0, Y = 0
Insert your move: 1
Current Board position :
X| .| .|
.| .| .|
.| .| .|

Current Board position :
X| .| .|
.| O| .|
.| .| .|

Evaluation time: 0.0299234s
Recommended move: X = 0, Y = 1
Insert your move: 3
Current Board position :
X| .| X|
.| O| .|
.| .| .|

Current Board position :
X| O| X|
.| O| .|
.| .| .|

Evaluation time: 0.0009975s
Recommended move: X = 2, Y = 1
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Recommended move: X = 2, Y = 1
Insert your move: 8
Current Board position :
X| O| X|
.| O| .|
.| X| .|

Current Board position :
X| O| X|
O| O| .|
.| X| .|

Evaluation time: 0.0009973s
Recommended move: X = 1, Y = 2
Insert your move: 6
Current Board position :
X| O| X|
O| O| X|
.| X| .|

Current Board position :
X| O| X|
O| O| X|
.| X| O|

Evaluation time: 0.0s
Recommended move: X = 2, Y = 0
Insert your move: 7
Current Board position :
X| O| X|
O| O| X|
X| X| O|

It's a tie!
PS D:\SEM_9\AI>
```