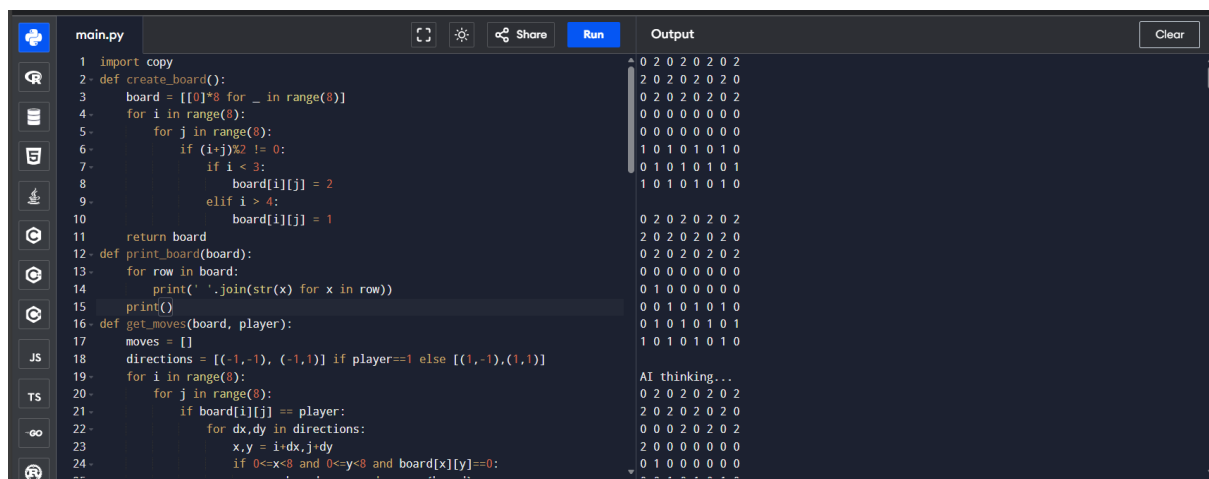


MLA0109 – Artificial Intelligence and Expert System

Python Programs – Day 2

1. Implement a Python program for a Checkers AI agent using the Minimax algorithm with Alpha–Beta pruning. The program should:
 - a) Represent the Checkers board and its legal moves.
 - b) Allow two players (Human vs AI) or (AI vs AI) to play.
 - c) Use Minimax with Alpha–Beta pruning for the AI's decision-making.
 - d) Display the board after every move and declare the winner at the end.



```
main.py
1 import copy
2 def create_board():
3     board = [[0]*8 for _ in range(8)]
4     for i in range(8):
5         for j in range(8):
6             if (i+j)%2 != 0:
7                 if i < 3:
8                     board[i][j] = 2
9                 elif i > 4:
10                    board[i][j] = 1
11    return board
12 def print_board(board):
13     for row in board:
14         print(' '.join(str(x) for x in row))
15     print()
16 def get_moves(board, player):
17     moves = []
18     directions = [(-1,-1), (-1,1)] if player==1 else [(1,-1),(1,1)]
19     for i in range(8):
20         for j in range(8):
21             if board[i][j] == player:
22                 for dx,dy in directions:
23                     x,y = i+dx,j+dy
24                     if 0<=x<8 and 0<=y<8 and board[x][y]==0:
25                         moves.append((i,j,x,y))
26     return moves
```

Output

```
0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0
0 2 0 2 0 2 0 2
0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0

0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0
0 2 0 2 0 2 0 2
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0

AI thinking...
0 2 0 2 0 2 0 2
2 0 2 0 2 0 2 0
0 0 2 0 2 0 2 0
2 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
- - - - -
```

2. Implement a Python program to simulate the Tic Tac Toe game. The program should:
 - a) Represent the 3×3 game board.
 - b) Allow a human player to play against an AI player.
 - c) Print the board after each move.
 - d) Declare the winner or draw at the end of the game

```

1 import random
2
3 def print_board(board):
4     print("\n")
5     for row in board:
6         print(" | ".join(row))
7         print("-" * 9)
8     print("\n")
9
10 def check_winner(board, p):
11     wins = [
12         [board[0][0], board[0][1], board[0][2]],
13         [board[1][0], board[1][1], board[1][2]],
14         [board[2][0], board[2][1], board[2][2]],
15         [board[0][0], board[1][0], board[2][0]],
16         [board[0][1], board[1][1], board[2][1]],
17         [board[0][2], board[1][2], board[2][2]],
18         [board[0][0], board[1][1], board[2][2]],
19         [board[0][2], board[1][1], board[2][0]],
20     ]
21     return [p,p,p] in wins
22
23 def empty_cells(board):
24     return [(i,j) for i in range(3) for j in range(3) if board[i][j] == " "]
25
26 def ai_move(board):
27     cells = empty_cells(board)
28     move = random.choice(cells)
29     board[move[0]][move[1]] = "O"
30
31 def play_demo():

```

STDIN

Input for the program (Optional)

```

x | |
-----
| x | o
-----
| |
-----

x | |
-----
| x | o
-----
| | x
-----

Human wins!

```

- Imagine you have a simple autonomous vacuum cleaner, which can move around in a grid-based environment. The grid is divided into cells, some of which are dirty, and some are clean. The vacuum cleaner can move in four directions: up, down, left, and right. Its primary task is to clean all the dirty cells while minimizing its movements. Implement the above AI problem using python.

```

1 import math
2
3 # Print the grid
4 def print_grid(grid, pos):
5     for i in range(len(grid)):
6         row = ""
7         for j in range(len(grid[0])):
8             if (i, j) == pos:
9                 row += "V " # Vacuum cleaner
10            elif grid[i][j] == 1:
11                row += "D " # Dirty cell
12            else:
13                row += "C " # Clean cell
14        print(row)
15    print()
16
17 # Manhattan distance between two points
18 def distance(p1, p2):
19     return abs(p1[0] - p2[0]) + abs(p1[1] - p2[1])
20
21 # Find closest dirty cell
22 def nearest_dirty_cell(grid, pos):
23     dirty_cells = [(i, j) for i in range(len(grid))
24                    for j in range(len(grid[0])) if grid[i][j] == 1]
25     if not dirty_cells:
26         return None
27     return min(dirty_cells, key=lambda d: distance(pos, d))
28
29 # Vacuum agent behavior
30 def vacuum_cleaner(grid, start):
31     pos = start

```

STDIN

Input for the program (Optional)

```

C C C
C V D
C D C

Cleaned cell at (1, 2)

C C C
C C V
C D C

C C C
C C C
C D V

Cleaned cell at (2, 1)

C C C
C C C
C V C

All cells cleaned!
Total steps: 7

```