

# **Artificial Intelligence**

## **Lab 6**

Submitted by

**Muhammad Bilal**

**FA21-BCS-110**



**Comsats University Islamabad,  
Abbottabad Campus.**

## Task 1 [10 points]

Write a Program to Implement Tic-Tac-Toe game using Python.

```
def print_board(board):
    """
    Prints the Tic-Tac-Toe board.
    """
    for row in board:
        print(" | ".join(row))
        print("-" * 5)

def check_winner(board, player):
    """
    Checks if the given player has won the game.
    """
    # Check rows
    for row in board:
        if all(cell == player for cell in row):
            return True

    # Check columns
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True

    # Check diagonals
    if all(board[i][i] == player for i in range(3)) or
all(board[i][2 - i] == player for i in range(3)):
        return True

    return False

def is_board_full(board):
    """
    Checks if the board is full (no empty cells).
    """
    for row in board:
        for cell in row:
            if cell == " ":
                return False
    return True

def main():
    """
```

```

Main function to run the Tic-Tac-Toe game.
"""
board = [["_ " for _ in range(3)] for _ in range(3)]
players = ["X", "O"]
turn = 0

while True:
    print_board(board)
    player = players[turn % 2]
    print(f"Player {player}'s turn")
    row = int(input("Enter row (0, 1, 2): "))
    col = int(input("Enter column (0, 1, 2): "))

    if board[row][col] == " ":
        board[row][col] = player
        if check_winner(board, player):
            print_board(board)
            print(f"Player {player} wins!")
            break
        if is_board_full(board):
            print_board(board)
            print("It's a tie!")
            break
        turn += 1
    else:
        print("That spot is already taken, try again.")

if __name__ == "__main__":
    main()

```

```

def print_board(board):
    for row in board:
        print(' '.join(row))
    print()

def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] and row[0] != ' ':
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return board[0][2]
    return None

# Game simulation
board = [' ']*9
row = 0
col = 0
turn = 'O'

print("Player O's turn")
row, col = input("Enter row (0, 1, 2): ").split()
row, col = int(row), int(col)
board[row*3+col] = 'O'

print("Player X's turn")
row, col = input("Enter row (0, 1, 2): ").split()
row, col = int(row), int(col)
board[row*3+col] = 'X'

print("Player O's turn")
row, col = input("Enter row (0, 1, 2): ").split()
row, col = int(row), int(col)
board[row*3+col] = 'O'

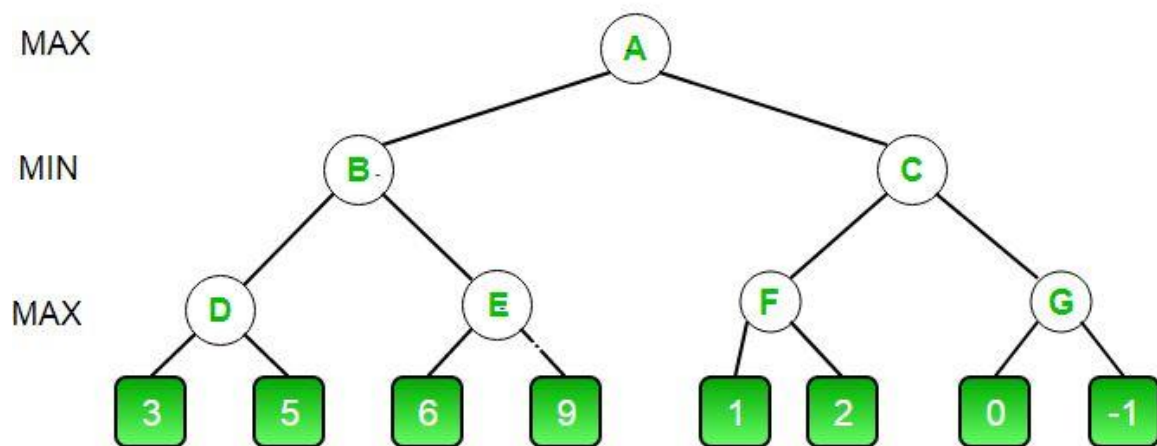
print("Player X's turn")
row, col = input("Enter row (0, 1, 2): ").split()
row, col = int(row), int(col)
board[row*3+col] = 'X'

print("Player X wins!")

```

## Task 2 [10 points]

Write a Program to Implement Alpha-Beta Pruning using Python.



```

def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

```

```

    if maximizingPlayer:
        best = float("-inf")
        for i in range(2):
            val = minimax(depth + 1, nodeIndex * 2 + i, False,
values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = float("inf")
        for i in range(2):
            val = minimax(depth + 1, nodeIndex * 2 + i, True,
values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best

if __name__ == "__main__":
    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is:", minimax(0, 0, True, values,
float("-inf"), float("inf")))

```

```

1  def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
2      if depth == 3:
3          return values[nodeIndex]
4
5      if maximizingPlayer:
6          best = float("-inf")
7          for i in range(2):
8              val = minimax(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
9              best = max(best, val)
10             alpha = max(alpha, best)
11             if beta <= alpha:
12                 break
13             return best
14      else:
15          best = float("inf")
16          for i in range(2):
17              val = minimax(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
18              best = min(best, val)
19              beta = min(beta, best)
20              if beta <= alpha:
21                  break
22              return best
23
24  if __name__ == "__main__":
25      values = [3, 5, 6, 9, 1, 2, 0, -1]
26      print("The optimal value is:", minimax(0, 0, True, values, float("-inf"), float("inf")))
27

```

PS C:\Users\Bilal\Desktop\New folder> python -u "c:\Users\Bilal\Desktop\New folder\alphaBeta.py"  
 The optimal value is: 5  
 PS C:\Users\Bilal\Desktop\New folder>