BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India

DEPARTMENT OF COMPUTER ENGINEERING
SUBJECT: Artificial Intelligence and Machine Learning

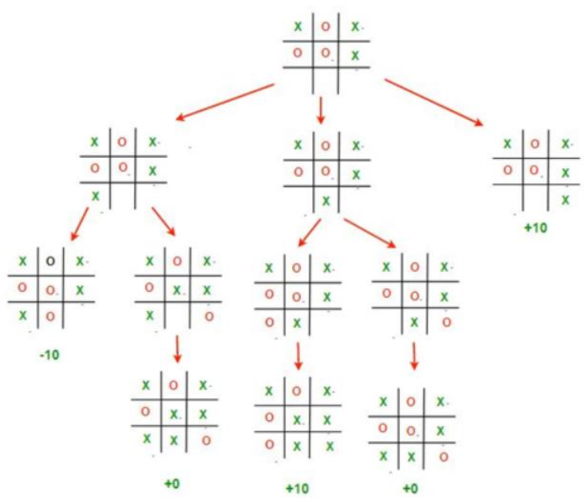
Name	Manish Shashikant Jadhav
UID no.	2023301005

Experiment 4

AIM :

Implement the problem using the Informed searching technique min-max algorithm . Analyze the algorithm with respect to Completeness, Optimality, time and space Complexity

a) Tic Tac Toe b) Implement Alpha Beta Pruning Algorithm on the same Game and put comparison



Tic-tac-toe using Min-Max Algorithm:

```
def print_board(board):
    for i in range(3):
        print(" ".join(board[i*3:(i+1)*3]))
    print()

def check_winner(board):
    # Check rows, columns and diagonals
    winning_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
        [0, 4, 8], [2, 4, 6]  # Diagonals
```

```
    ]
    for combo in winning_combinations:
        if board[combo[0]] == board[combo[1]] == board[combo[2]] != ' ':
            return board[combo[0]]
    if ' ' not in board:
        return 'Tie'
    return None

def minimax(board, depth, is_maximizing):
    result = check_winner(board)
    if result == 'X':
        return 1
    elif result == 'O':
        return -1
    elif result == 'Tie':
        return 0

    if is_maximizing:
        best_score = float('-inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = minimax(board, depth + 1, False)
                board[i] = ' '
                best_score = max(score, best_score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                score = minimax(board, depth + 1, True)
                board[i] = ' '
                best_score = min(score, best_score)
        return best_score

def best_move(board):
    best_score = float('-inf')
```

```
    move = -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            score = minimax(board, 0, False)
            board[i] = ' '
            if score > best_score:
                best_score = score
                move = i
    return move

def play_game():
    board = [' ' for _ in range(9)]
    print("Initial board:")
    print_board(board)

    while True:
        move = best_move(board)
        board[move] = 'X'
        print("AI's move:")
        print_board(board)

        if check_winner(board):
            break

        player_move = int(input("Enter your move (0-8): "))
        board[player_move] = 'O'
        print("Your move:")
        print_board(board)

        if check_winner(board):
            break

    result = check_winner(board)
    if result == 'X':
        print("AI wins!")
    elif result == 'O':
        print("You win!")
```

```
    else:
        print("It's a tie!")

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

Output:

```
PS D:\Manish\SPIT> & C:/Users/manis/AppData/Local/Programs/Python/P
Initial board:




AI's move:
X




Enter your move (0-8): 1
Your move:
X O




AI's move:
X O
X


Enter your move (0-8): 6
Your move:
X O
X
O

AI's move:
X O
X X
O

Enter your move (0-8): 5
Your move:
X O
X X O
O

AI's move:
X O
X X O
O   X

AI wins!
PS D:\Manish\SPIT>
```

Tic-tac-toe using Alpha beta pruning:

```
def print_board(board):
    for i in range(3):
        print(" ".join(board[i*3:(i+1)*3]))
    print()

def check_winner(board):
    winning_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
        [0, 4, 8], [2, 4, 6]  # Diagonals
    ]
    for combo in winning_combinations:
        if board[combo[0]] == board[combo[1]] == board[combo[2]] != ' ':
            return board[combo[0]]
    if ' ' not in board:
        return 'Tie'
    return None

def alpha_beta(board, depth, alpha, beta, is_maximizing):
    result = check_winner(board)
    if result == 'X':
        return 1
    elif result == 'O':
        return -1
    elif result == 'Tie':
        return 0

    if is_maximizing:
        best_score = float('-inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = alpha_beta(board, depth + 1, alpha, beta, False)
                board[i] = ' '
                best_score = max(score, best_score)
                alpha = max(alpha, best_score)
                if beta <= alpha:
                    break
```

```
        return best_score
    else:
        best_score = float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                score = alpha_beta(board, depth + 1, alpha, beta, True)
                board[i] = ' '
                best_score = min(score, best_score)
                beta = min(beta, best_score)
                if beta <= alpha:
                    break
        return best_score

def best_move(board):
    best_score = float('-inf')
    move = -1
    alpha = float('-inf')
    beta = float('inf')
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'X'
            score = alpha_beta(board, 0, alpha, beta, False)
            board[i] = ' '
            if score > best_score:
                best_score = score
                move = i
    return move

def play_game():
    board = [' ' for _ in range(9)]
    print("Initial board:")
    print_board(board)

    while True:
        move = best_move(board)
        board[move] = 'X'
        print("AI's move:")
```

```
        print_board(board)

        if check_winner(board):
            break

        player_move = int(input("Enter your move (0-8): "))
        board[player_move] = 'O'
        print("Your move:")
        print_board(board)

        if check_winner(board):
            break

    result = check_winner(board)
    if result == 'X':
        print("AI wins!")
    elif result == 'O':
        print("You win!")
    else:
        print("It's a tie!")

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

Output:

```
PS D:\Manish\SPIT> & C:/Users/manis/AppData/Local/Programs
Initial board:




AI's move:
X



Enter your move (0-8): 8
Your move:
X

    O

AI's move:
X   X

    O

Enter your move (0-8): 1
Your move:
X O X

    O

AI's move:
X O X

X   O

Enter your move (0-8): 3
Your move:
X O X
O
X   O

AI's move:
X O X
O X
X   O

AI wins!
PS D:\Manish\SPIT>
```

Analysis of Algorithm

- **Completeness:** Both minimax and alpha-beta pruning are complete for Tic-Tac-Toe. They explore the entire game tree and will always find a solution if one exists.
- **Optimality:** Both algorithms are optimal for Tic-Tac-Toe. They will always find the best possible move, assuming perfect play from both sides.

	• **Time Complexity:** 1. **Minimax:** O(b^d), where b is the branching factor (9 at the start, decreasing as the game progresses) and d is the maximum depth of the tree (9 for Tic-Tac-Toe). 2. **Alpha-Beta Pruning:** O(b^(d/2)) in the best case, but still O(b^d) in the worst case. On average, it's much faster than minimax. • **Space Complexity:** Both algorithms have a space complexity of O(d), where d is the maximum depth of the tree. This is due to the recursive nature of the algorithms, which use the call stack.
Comparison:	• **Functionality:** Both algorithms produce the same optimal results for Tic-Tac-Toe. • **Performance:** Alpha-beta pruning is generally faster, especially for larger game trees, as it can prune significant portions of the tree. For Tic-Tac-Toe, the difference may not be as noticeable due to the small game tree. • **Implementation:** Alpha-beta pruning is slightly more complex to implement but offers performance benefits.
CONCLUSION:	Hence by completing this experiment I came to know about Informed searching technique min-max algorithm and alpha-beta pruning.