Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

Date: 7/10/2024

EXPERIMENT NO.: 05

AIM: Designing and Implementing a Tic-Tac-Toe Game for Human-Computer Interaction.

OBJECTIVES: From this experiment, it will be able to:

- Understand Alpha Beta Pruning in AI.

THEORY:

1. Introduction to Tic-Tac-Toe:
Tic-Tac-Toe is a simple two-player game played on a 3x3 grid. Players take turns marking a square with their respective symbols (typically 'X' and 'O'). The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. The game is often used as a fundamental example to demonstrate concepts in game theory and artificial intelligence (AI), especially with respect to decision-making and optimization algorithms.

2. Game Representation:
The Tic-Tac-Toe board can be represented as a 3x3 matrix.
Players alternate turns. Player 1 (human) marks an 'X', while Player 2 (computer) marks an 'O'.
The goal is to either win by making a line of three identical marks or to prevent the opponent from winning.

3. Minimax Algorithm
The Minimax algorithm is a recursive decision-making process used in two-player games. The primary aim is to minimize the possible loss while maximizing the potential gain. In Tic-Tac-Toe, the computer assumes that the human player will always make the best possible move.

- Min player (Human): Tries to minimize the computer's score.
- Max player (Computer): Tries to maximize the computer's score.

At each node (or game state), the Minimax algorithm evaluates all possible moves, recursively calculating the optimal strategy until it reaches a terminal state (win, loss, or draw).

4. Alpha-Beta Pruning
Alpha-Beta Pruning is an optimization technique for the Minimax algorithm. It reduces the number of nodes evaluated by the algorithm, effectively improving its efficiency by pruning branches that do not need to be explored.

- Alpha: The best value that the maximizer (computer) can guarantee at that point or better.
- Beta: The best value that the minimizer (human) can guarantee at that point or better.

During the evaluation, if the computer (Max player) finds a move that is worse than a previously examined move (from the perspective of the human), further exploration of that branch is unnecessary (pruning). Similarly, if the human (Min player) finds a move worse than what has already been considered for the computer, the remaining nodes in that branch are skipped.

This leads to:
Time Complexity Improvement: From O(b^d) in regular Minimax (where `b` is the branching factor and `d` is the depth of the game tree) to approximately O(b^(d/2)) in Alpha-Beta Pruning.

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

Date: 7/10/2024

5. Game Tree Construction

In Tic-Tac-Toe, every possible state of the game can be represented as a node in a tree. The root node represents the initial state of the game (empty board). Each child node represents a possible move by a player. The Minimax algorithm, with Alpha-Beta Pruning, evaluates all possible game states to determine the optimal move for the computer.

6. Implementation of Alpha-Beta Pruning in Tic-Tac-Toe

I. Initial Setup: The game starts with an empty board. The computer and the human player alternate turns.
Evaluation Function: This function evaluates the state of the game. It assigns scores based on the following conditions:
- +10 if the computer wins.
- -10 if the human wins.
- 0 if the game is a draw.

II. Recursive Call: The algorithm recursively explores each possible move, updating the alpha and beta values based on the results.
- If the current state results in a win for the computer, the recursion returns a positive score.
- If it results in a win for the human, the recursion returns a negative score.

III. Pruning Condition: If the current move yields a value less than or greater than the current alpha or beta, the remaining branches of that subtree are pruned, i.e., not further explored.

7. Human-Computer Interaction

The interaction in this experiment is based on the computer responding to human moves using the Alpha-Beta Pruning algorithm. After each human move, the AI evaluates the board, decides the best possible response move by exploring the game tree, and makes its move.

The interaction design involves:
- Input Mechanism: The human selects a square to place their mark.
- AI Response: After evaluating the board, the AI chooses the optimal move and places its mark.
- Feedback: The game board is updated, displaying the current state after each turn.
- Endgame Detection: The game constantly checks if either the human or the computer has won or if the game has reached a draw state.

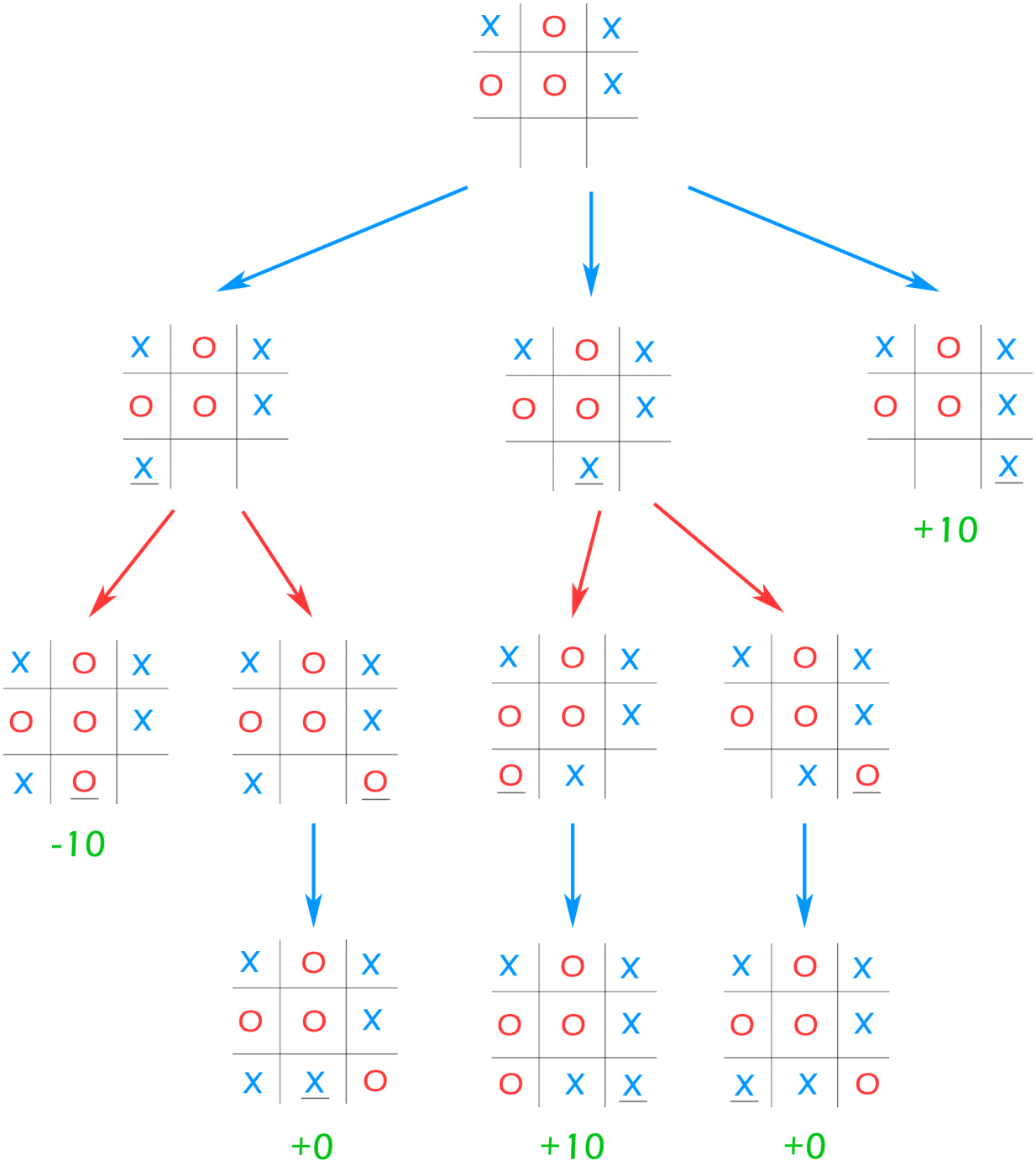
8. Advantages of Alpha-Beta Pruning in Tic-Tac-Toe

- Efficiency: The game tree for Tic-Tac-Toe has a branching factor of about 9 at the beginning (as each player has 9 possible moves initially), but as the game progresses, this decreases. Alpha-Beta Pruning ensures that not all possibilities are explored, reducing unnecessary computations.
- Optimal Gameplay: The use of Alpha-Beta Pruning ensures that the computer makes optimal moves quickly without sacrificing performance.

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

Date: 7/10/2024

EXAMPLE: Tic-Tac-Toe Game for Human-Computer Interaction.



PROGRAM

```
# Tic-Tac-Toe Game using Alpha-Beta Pruning
HUMAN, COMPUTER = 'X', 'O'
board = [' ' for _ in range(9)]
def print_board():
    for row in [board[i:i + 3] for i in range(0, 9, 3)]:
        print('|' + '|'.join(row) + '|')
def is_moves_left(board):
```

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

Date: 7/10/2024

```
    return ' ' in board

def evaluate(b):
    for row in range(0, 9, 3):
        if b[row] == b[row+1] == b[row+2] != ' ': return 10 if b[row] == COMPUTER else -10
    for col in range(3):
        if b[col] == b[col+3] == b[col+6] != ' ': return 10 if b[col] == COMPUTER else -10
    if b[0] == b[4] == b[8] != ' ': return 10 if b[0] == COMPUTER else -10
    if b[2] == b[4] == b[6] != ' ': return 10 if b[2] == COMPUTER else -10
    return 0

def minimax(board, depth, is_maximizing, alpha, beta):
    score = evaluate(board)
    if score == 10 or score == -10: return score - depth if score == 10 else score + depth
    if not is_moves_left(board): return 0
    if is_maximizing:
        best = -float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = COMPUTER
                best = max(best, minimax(board, depth+1, False, alpha, beta))
                board[i] = ' '
                alpha = max(alpha, best)
                if beta <= alpha: break
        return best
    else:
        best = float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = HUMAN
                best = min(best, minimax(board, depth+1, True, alpha, beta))
                board[i] = ' '
                beta = min(beta, best)
                if beta <= alpha: break
        return best

def find_best_move():
    best_value, best_move = -float('inf'), -1
    for i in range(9):
        if board[i] == ' ':
            board[i] = COMPUTER
            move_value = minimax(board, 0, False, -float('inf'), float('inf'))
            board[i] = ' '
            if move_value > best_value:
                best_move = i
                best_value = move_value
```

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

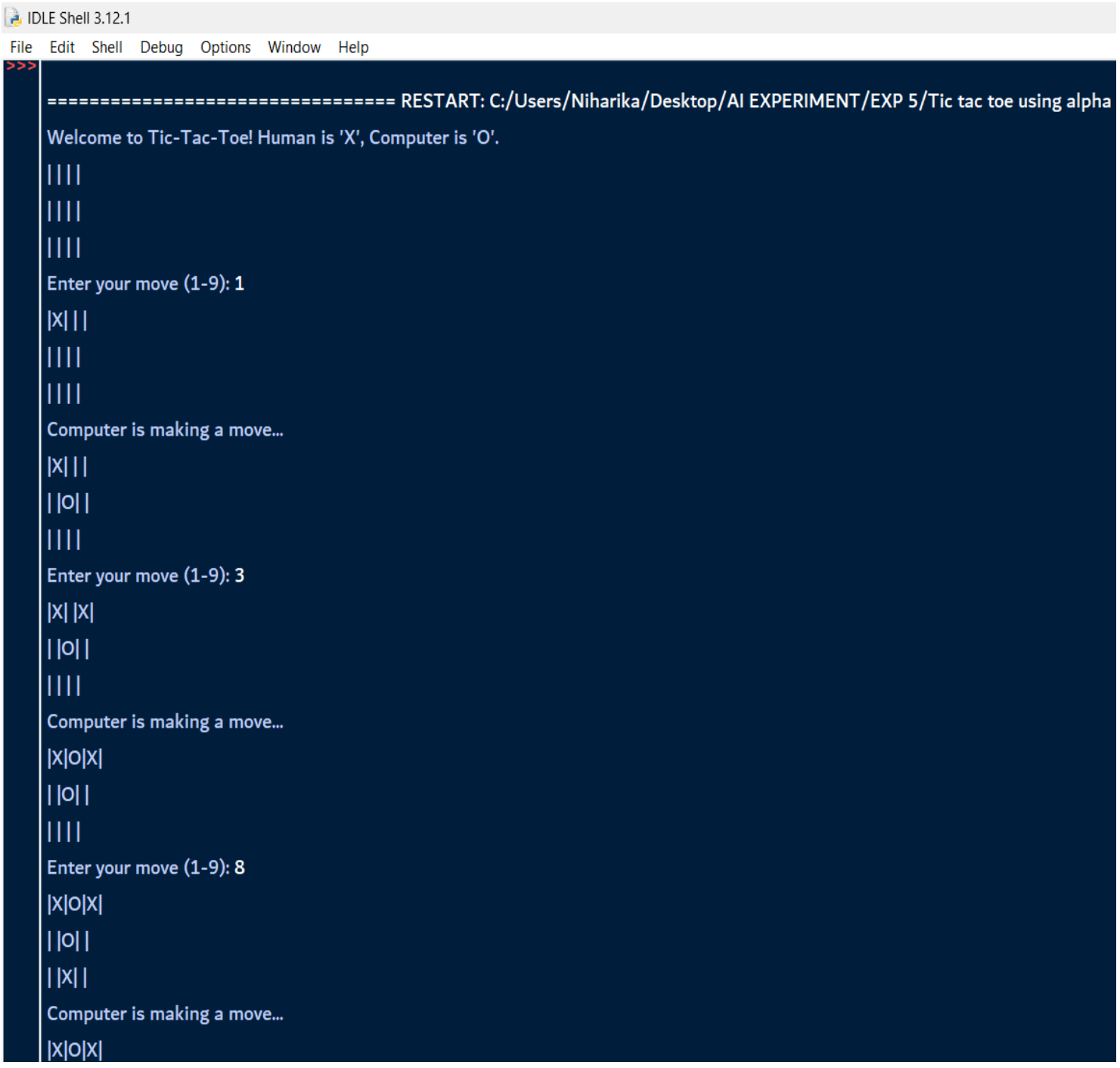
Date: 7/10/2024

```
    return best_move
def check_game_status():
    score = evaluate(board)


if score == 10: print_board(); print("Computer wins!"); return True
    elif score == -10: print_board(); print("Human wins!"); return True
    elif not is_moves_left(board): print_board(); print("It's a draw!"); return True
    return False

def play_game():
    print("Welcome to Tic-Tac-Toe! Human is 'X', Computer is 'O'.")
    print_board()
    while True:
        # Human's move
        while True:
            human_move = int(input("Enter your move (1-9): ")) - 1
            if board[human_move] == ' ':
                board[human_move] = HUMAN
                break
            else: print("Invalid move! Try again.")
        print_board()
        if check_game_status(): break
        # Computer's move
        print("Computer is making a move...")
        computer_move = find_best_move()
        board[computer_move] = COMPUTER
        print_board()
        if check_game_status(): break
play_game()
```

Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

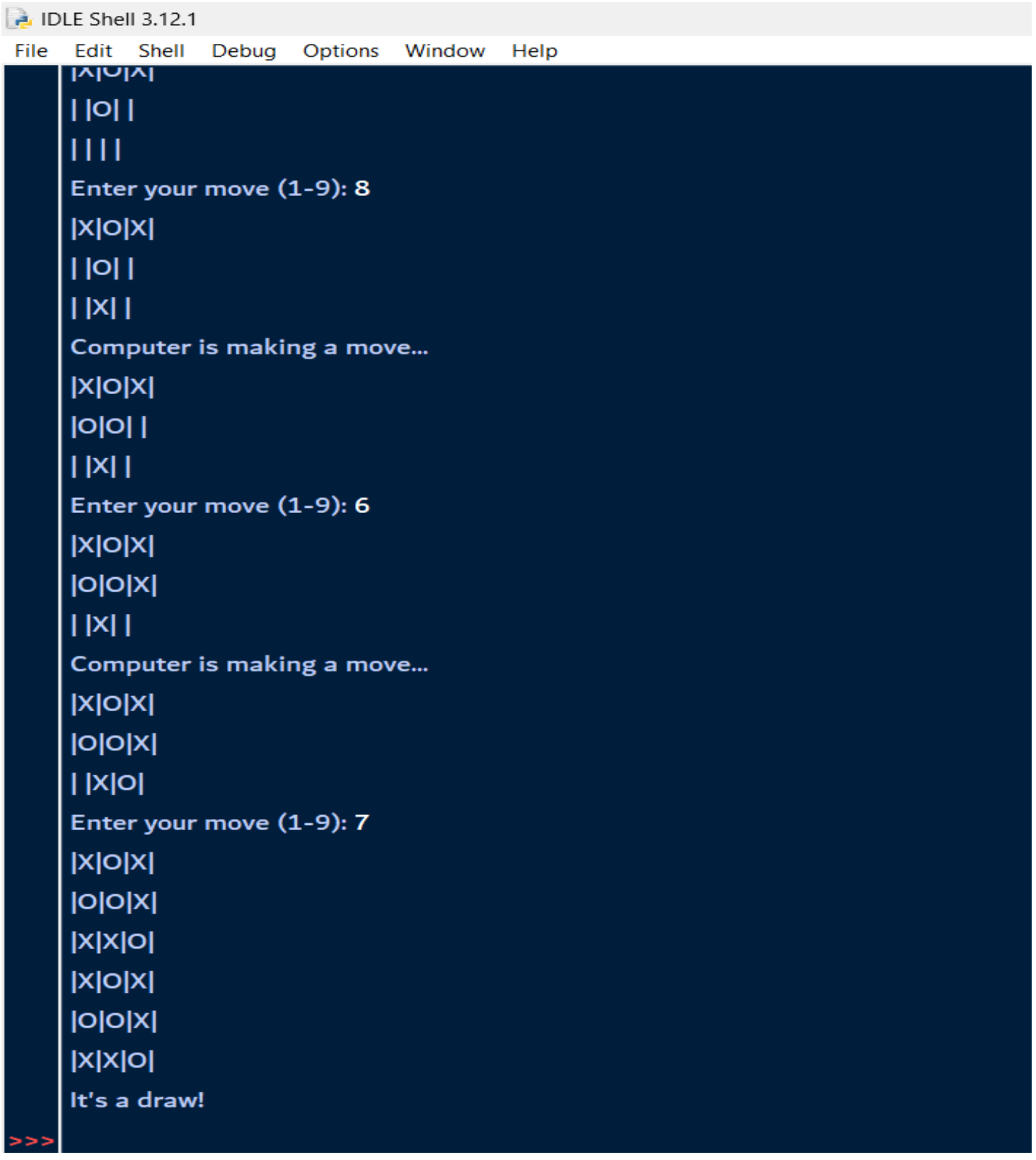
Date: 7/10/2024

RESULTS



Class and Branch: TE AI & DS
Student Roll no.: A07
Student Name:Niharika Seth

Date: 7/10/2024



CONCLUSION

In conclusion, using Alpha-Beta Pruning in a Tic-Tac-Toe game enhances the performance of the AI by pruning unproductive branches in the decision tree, ensuring a faster and more efficient solution to the game. The interaction between the human and the computer demonstrates the power of decision-making algorithms in real-time gaming environments, providing insights into the broader application of such AI techniques in human-computer interaction scenarios.

****