SSN College of Engineering, Kalavakkam

Department of Computer Science and Engineering

IV Semester – B.E (CSE)

R2024

UCS3461 - Foundations of Artificial Intelligence (TCP-Lab)

Academic Year: 2025-2026 Even **Batch: 2024-2028**

Exercise 3 Tic-Tac-Toe Game using Minimax and Alpha-Beta Pruning Search Methods (CO2, K5, PIs: 1.3.1,1.4.1,2.1.3,3.1.1,4.2.1,12.3.2,12.4.1)

Design and implement a Tic-Tac-Toe game using the Minimax algorithm with Alpha–Beta Pruning by representing the entire game within a single class named **TicTacToeGame.**

Class Specification: TicTacToeGame

The *TicTacToeGame* class represents the complete Tic-Tac-Toe game environment and the AI decision-making logic. It maintains the current board configuration as the game state and provides methods for state generation, game evaluation, and optimal move selection using adversarial search.

Data Members

The class contains a data member *board* to store the current Tic-Tac-Toe board configuration and a data member *current_player* to indicate whose turn it is during the game.

Methods

- The method *display_board()* is used to display the current state of the Tic-Tac-Toe board in a clear and readable format for the player.
- The method *generate_successor_states()* generates all possible valid successor states from the current game state by placing the current player's mark in each available empty position. These successor states are used to expand the game tree during search.
- The method *is_moves_left()* checks whether there are any legal moves remaining in the current state of the game.
- The method *check_winner()* examines the board configuration to determine whether the AI player or the human player has achieved a winning combination along any row, column, or diagonal.
- The method *is_terminal()* determines whether the current game state is a terminal state, either because one of the players has won or because no further moves are possible, resulting in a draw.
- The method *utility()* implements the utility function for terminal states of the game. It returns +1 if the AI player wins, −1 if the human player wins, and 0 if the game ends in a draw.
- The method *minimax()* implements the Minimax algorithm by recursively evaluating all successor states, alternating between maximizing and minimizing players, and propagating utility values upward in the game tree.
- The method *alpha_beta()* enhances the Minimax algorithm by applying Alpha–Beta Pruning to eliminate branches of the game tree that cannot influence the final decision, thereby improving computational efficiency.

- The method *find_best_move()* evaluates all successor states from the current position using Minimax with Alpha–Beta Pruning and selects the optimal move for the AI player.
- The method *play_game()* controls the overall flow of the game by alternating turns between the human player and the AI, updating the board after each move, and terminating the game when a win or draw condition is reached.

Compare Minimax and Alpha–Beta Pruning

- Compare the number of game states (nodes) explored by the Minimax algorithm and the Alpha–Beta Pruning algorithm for the same Tic-Tac-Toe game configuration. Explain why there is a difference.
- Analyze the time complexity of Minimax and Alpha–Beta Pruning for Tic-Tac-Toe. How does Alpha–Beta Pruning improve efficiency without affecting the final decision?
- Explain whether Alpha–Beta Pruning changes the optimal move selection compared to Minimax. Justify your answer.