Meeting Minutes

Meeting Minutes: Tic-Tac-Toe Game Implementation Review

Date: [Date of Code Creation - Assuming Today]

Attendees: Krishna Thakkar (C075, 60004220, 250)

Subject: Review of Tic-Tac-Toe Game Implementation using Minimax Algorithm

Key Discussion Points:

Initialization:

The code initializes the game board as a list of 9 spaces, representing the Tic-Tac-Toe grid.

`board = [' ' for _ in range(9)]` creates an empty board.

Board Printing:

The `print_board()` function formats and displays the current state of the Tic-Tac-Toe board in the console.

It iterates through the board in rows of three, creating a visually clear representation.

Winning Condition Check:

`check_winner(player)` determines if a player ('X' or 'O') has won the game.

It checks all possible winning combinations: rows, columns, and diagonals.

`win_conditions` list stores these combinations as tuples of indices.

Tie Condition Check:

`is_board_full()` checks if all spaces on the board are occupied, indicating a tie.

It verifies that there are no empty spaces (' ') remaining on the board.

Available Moves:

`available_moves()` returns a list of indices representing the empty spaces on the board.

This list is used by the Minimax algorithm to explore possible moves.

Minimax Algorithm:

`minimax(is_maximizing)` implements the Minimax algorithm to determine the optimal move for the AI.

It recursively explores the game tree, assigning scores to each possible outcome.

`is_maximizing` parameter indicates whether the AI is trying to maximize its score or minimize the opponent's score.

Base cases:

AI wins: returns 1

Human wins: returns -1

Tie: returns 0

Recursive step:

If maximizing, it iterates through available moves, simulates the AI making the move, calls minimax recursively to get the score for the opponent's best move, and updates the `best_score` to the maximum of the current `best_score` and the returned score.

If minimizing, it does the same, but updates the `best_score` to the minimum of the current `best_score` and the returned score.

AI Move Selection:

`ai_move()` uses the Minimax algorithm to choose the best move for the AI.

It iterates through available moves, calls minimax to evaluate the score of each move, and selects the move with the highest score.

Player Move Handling:

`player_move()` prompts the player to enter their move (1-9).

It validates the input and updates the board with the player's move ('X').

Handles invalid moves by prompting the player to try again.

Game Over Check:

`is_game_over()` checks if the game has ended (either a player has won or the board is full).

It calls `check_winner()` for both players and `is_board_full()`.

Main Game Loop:

`play_game()` contains the main game loop.

It prints the welcome message and the initial board.

It alternates between player and AI moves until the game is over.

It prints the result of the game (win, lose, or tie).

Decisions Made:

The Tic-Tac-Toe game will be implemented using the Minimax algorithm for AI decision-making.

The board will be represented as a list of 9 elements.

The AI will be represented by 'O' and the player by 'X'.

The game will continue until a player wins or the board is full.

Action Items:

No action items were identified in the provided code. The code represents a complete implementation of the Tic-Tac-Toe game.