CMPT 310 Assignment 2 - TicTacToe

Jiaqing Hu (Vincent)

GameState Class

- **to_move**: The player who is supposed to make the next move (e.g., 'X' or 'O').
- **move**: The last move that led to this state.
- **utility**: The utility score of this state for a given player (1 for win, -1 for loss, 0 otherwise).
- **board**: A dictionary representing the game board, with keys as positions and values as the player occupying that position.
- **moves**: A list of possible moves that the current player can make.

Game Class

- **actions**: Returns the available moves for a state.
- **result**: Returns the next state after a move.
- **utility**: Returns the utility of a terminal state for a player.
- **terminal_test**: Tests whether the game has ended (either win/loss or a draw).
- **to_move**: Returns which player is to move.

Tic-Tac-Toe Game Subclass

- **result**: Updates the game state after a move is made.
- **terminal_test**: Determines if the game is over by checking if there are no available moves or if there is a winner.
- **compute_utility**: Calculates the utility of the game state (win/loss).
- **k** : number of consecutive pieces required to win the game
- **maxDepth** : max depth possible of the board
- **switchPlayer()** : switch current player
- The game board is represented as a dictionary where keys are positions (tuples) and values are players ('X' or 'O').

Minimax

- **Goal**: To determine the best move by exploring all possible future moves recursively

- **max_value**: Tries to maximize the score for the current player.
- **min_value**: Tries to minimize the score for the opponent.

Alpha-Beta Pruning

- A more efficient version of the Minimax algorithm, *alpha_beta()*
- **Goal**: It reduces the number of nodes by pruning

- **alpha**: The best value that the **maximizer** currently can guarantee. It represents a **lower bound** on the possible outcomes for the maximizing player.
- **beta**: The best value that the **minimizer** currently can guarantee. It represents an **upper bound** on the possible outcomes for the minimizing player.

Evaluation Function

- used to **estimate** the value of non-terminal states. (in cutoff functions)

- **eval1**: Evaluates the game state by counting the number of almost complete rows/columns/diagonals (i.e., k-1 matches).

Monte Carlo Tree Search (MCTS)

- find the optimal move by simulating games.

- balances **exploration** (trying out **less certain moves**) and **exploitation** (focusing on known **successful moves**).

Monte Carlo Tree Search (MCTS)

- ***monteCarloPlayer(self, timelimit = 4)***
- **Selection**: Traverse the tree from the root to a leaf node by selecting the node with the highest UCT (Upper Confidence Bound for Trees) value.
- **Expansion**: If the selected node is not a terminal state, expand the node by adding its child nodes.
- **Simulation**: Simulate random play from the new node until a **terminal state** (win/loss/draw) is reached.
- **Backpropagation**: Update the **win scores** and visit counts from the leaf node back up to the root, adjusting the tree based on the result of the simulation.

Monte Carlo Tree Search (MCTS)

- ***selectNode(self, nd)***
- Starting from the root node, this function repeatedly chooses the child with the highest UCT value; ***findBestNodeWithUCT(), uctValue()***
- Stop when it reaches a **leaf node** (a node with no children);
- Return the leaf node for expansion.

Monte Carlo Tree Search (MCTS)

- ***expandNode(self, nd)***
- Generates new child nodes for the selected node;
- Apply all possible legal actions;
- These child nodes represent the game states that result from making each possible move.

Monte Carlo Tree Search (MCTS)

- ***simulateRandomPlay(self, nd)***
- From a leaf node, this function plays out a random game to a terminal state (win/loss/draw).
- Simulate moves until a winner is found or draw(random playout).
- Before starting the random play, check if the current node represents a **winning state** for the **opponent**.
- If so, mark the score accordingly.

Monte Carlo Tree Search (MCTS)

- ***backPropagation(self, nd, winningPlayer)***
- Once reaches a terminal state, propagate the result back up the tree.
- Increase ***visitCount***
- Update ***winScore*** if ***winningPlayer*** matches.