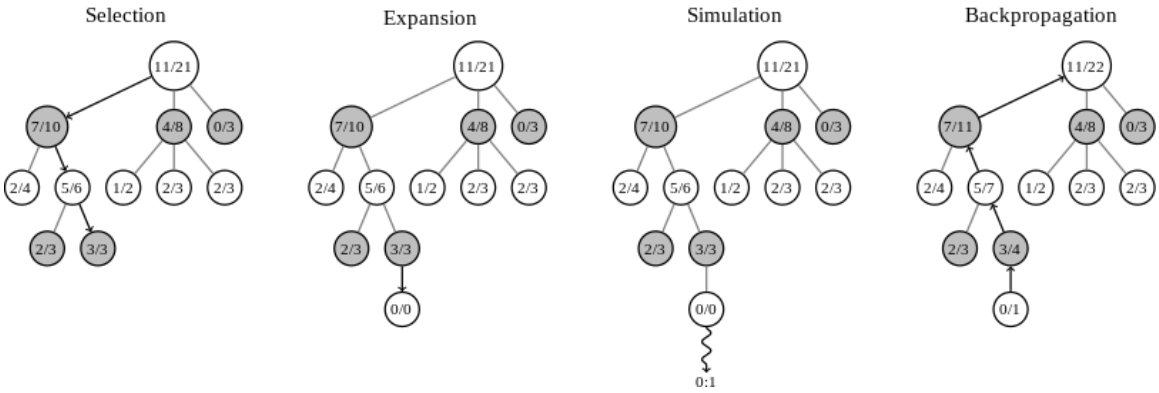
Monte-Carlo Tree Search

Monte-Carlo tree search is a decision making algorithm that is based on random sampling and is heavily used in game trees.

MCTS algorithm

The MCT algorithm explores the most promising nodes based on random sampling for all the child nodes of the current node. Each child node has a score which estimates how promising node could be. For each child node, a random tree decision is taken until reaching a leaf node. The output of each simulation is used to update the estimation of the corresponding child node score. For a large number of simulations, the MCTS converges the minimax evaluation.



Advantages and Disadvantages

The basic version of MCTS takes a large number of simulations to converge to the minimax evaluation. However, MCTS doesn't not need a game-specific evaluation. Only the game mechanics are enough to be implemented. This makes MCTS compatible with any game with finite states without the need to develop any kind of game specific rules. Also, MCTS' running time can be controlled by the developer which is critical in real-time tree search.

Design

The implemented version of MCTS is the basic version. After reading the input, `MonteCarloTS` is called.

`MonteCarloTS` gets all the valid moves `(validMoves)`. For each valid move, `simulateGame` is called. All simulations are called multiple times until the 3000ms pass. Each `simulateGame` returns 2 outputs; the winner and how many moves counted until the win occurred. For each result, the output is given to the `moveResults` array which counts how many wins, losses, and ties this valid move has gathered so far. It also holds `weightedLosses` and `weightedWins` which favors less number of moves to win/lose. After the 3000ms had passed, a score for each valid move is calculated. The calculation is a weighted score of each win and loss. The valid move with the highest score is chosen.

$$weight = 10 * maximum\ remaining\ moves - simulation\ moves$$

$$weightedWin/Loss = weight * win/loss$$

$$score = \frac{weightedWins - weighedLosses}{wins + losses + ties}$$

Game mechanics functions

`simulateGame`

`simulateGame` simulates the game with random moves each time for both players until the game ends. All other functions are used to implement the Ultimate Tic Tac Toe game mechanics.

`gameRemainingMoves`

Returns the maximum remaining moves until the game is tied. Essentially returns the count of all empty spaces.

`getGlobalValidMoves`

Returns the valid moves in case the player is directed to play in an already owned/tied square.

gameOver

Returns true if the game is tied.

checkSquareWinner

Returns a character representing the winner of a square. Returns '-' if the square is not wined or tied and returns '!' if the square is tied.

checkRow (row, board, squareLocation)

This function is called in `checkSquareWinner` and returns the winner of the input row. Returns '-' if the row isn't won by a player.

checkSquareTie

This function is called in `checkSquareWinner` and checks if the square is tied. Returns true if there is a tie.

checkcolumn (column, board, squareLocation)

This function is called in `checkSquareWinner` and returns the winner of the input column. Returns '-' if the column isn't won by a player.

checkGameWinner

This function is called in `simulateGame` which returns the winner of the game. Returns '-' if the game isn't won by a player. Its input is the `SquareWinner` board.

getBoardCopy

This function is used in `MonteCarloTS` for each `simulateGame` instance. Returns a copy of the playing board.

getSquareWinnerCopy

This function is used in `MonteCarloTS` for each `simulateGame` instance. Returns a copy of the winner of each square.

checkRow (row, board)

This function is called in `checkGameWinner` and returns the winner of the input row. Returns '-' if the row isn't won by a player.

checkcolumn (column, board)

This function is called in `checkGameWinner` and returns the winner of the input column. Returns '-' if the column isn't won by a player.

getValidMoves

This function is called in `MonteCarloTS` and `simulateGame` and returns all the valid moves.