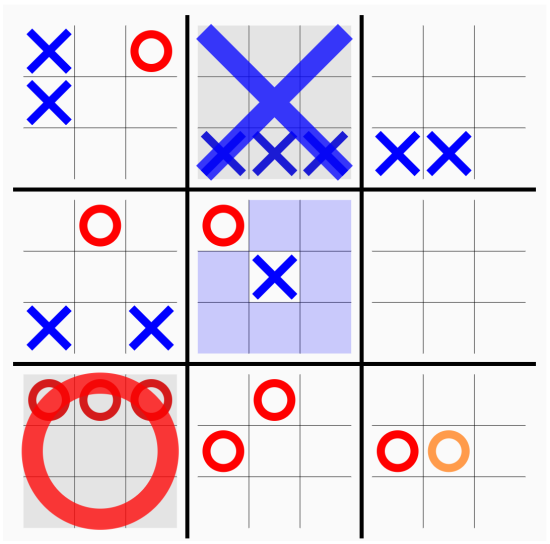
UTTTSolver: Ultimate Tic-Tac-Toe Solver Project Proposal

Fernando Macchiavello Cauvi (FM2758), Hasan Alqassab (HA2613)

Overview:

Context Ultimate Tic-Tac-Toe (UTTT) is a variant of the popular game Tic-Tac-Toe. UTTT is set up by having a 3x3 Tic-Tac-Toe board with each cell containing an interior Tic-Tac-Toe board:



Rules "Just like in regular tic-tac-toe, the two players (X and O) take turns, starting with X. The game starts with X playing wherever they want in any of the 81 empty spots. Next the opponent plays, however they are forced to play in the small board indicated by the relative location of the previous move. For example, if X plays in the top right square of a small (3 × 3) board, then O has to play in the small board located at the top right of the larger board. Playing any of the available spots decides in which small board the next player plays.

If a move is played so that it is to win a small board by the rules of normal tic-tac-toe, then the entire small board is marked as won by the player in the larger board. Once a small board is won by a player or it is filled completely, no more moves may be played in that board. If a player is sent to such a board, then that player may play in any other board. Game play ends when either a player wins the larger board or there are no legal moves remaining, in which case the game is a draw." (Orlin, Ben (June 1, 2013). "Ultimate Tic-Tac-Toe". *Math with Bad Drawings*.)

Project Goals

1. Create a fully functional Haskell implementation of Ultimate Tic-Tac-Toe that is playable from within the terminal.
2. Implement a parallelized solver in Haskell (more details below) that runs to play against (and hopefully beat) the user.

Implementation:

In this two-player zero-sum game, the player and the AI agent are in direct competition, each striving to win the game. The AI agent aims to maximize the objective function, while the player aims to minimize it, having no shared interests between them. Typically, such games are effectively solved using either the Minimax algorithm or the Monte Carlo Tree Search (MCTS) algorithm.

Minimax Algorithm with alpha beta pruning: This is a recursive strategy used in turn-based games, assuming both players play optimally. It involves creating a game tree where each node represents a game state. The algorithm uses heuristics to assign values to these states, with high values favoring the AI. The AI's goal is to maximize its score, anticipating that the human player will try to minimize it. This approach is ideal for games with fewer possible moves, but it becomes computationally intensive for games with larger search spaces such as this game. Since we cannot traverse the entire decision tree (as there are more than 5^45 nodes) we can stop the search at a specific depth and use the evaluation of the states. The implementation of this algorithm is straightforward. However, the difficult part will be coming up with the heuristics.

Monte Carlo Tree Search (MCTS): MCTS is a more modern approach that uses randomness to simulate game outcomes. It builds a search tree by randomly sampling moves in the game, then uses the results of these simulations to estimate the most promising moves. MCTS is particularly effective in games with large search spaces where traditional exhaustive search methods like Minimax are impractical. It balances between exploring new, potentially promising moves and exploiting known, successful strategies.

While further exploring the intricacies of creating a UTTT solver, we will end up deciding which of these approaches suits the problem best, and move forward with it. This step has not been decided on yet and will be chosen with more research on the subject. Both of these approaches lend themselves easily to a parallelized approach and will be needed to create an efficient algorithm.