Connect Fourge README

Team Members: *Spencer Dellenbaugh (sdellenb), Jusung Lee (jlee418), Sidney Schwartz (sschwa11)*

Fixed Model Structure (first attempt): Mimics the TicTacToe model explained in lecture with fixed indices. Uses a filler player and a dummy bottom index to simulate gravity in the model. This model grew frustrating, as board dimension had to be hard-coded, so we followed some of Izzy's advice and worked to develop an extensible model instead, which can be run for any number of rows and columns (within reason). The final, extensible model is detailed below.

Extensible Model Structure (our final submission):

Sigs:

- Row, Column: These are each connected in a chain-like structure of 'previous' and 'next' relations to allow a representation of order and adjacency
- Board: stores p1 and p2 relations to represent two players, each of which stores a set of relations in the form Row→Column to mark that player's moves

Preds:

- wellformed: Ensures the 'prev' and 'next' relations are properly formed for Row and Column. One initial and one final Row and Column that have no previous and next respectively. Any two distinct Row/Column are transitively connected by either the prev or next relation. Prev and next must be the transpose of each other. Lastly, ensures both players cannot make the same move.

Turns: One of these must be true for the board to be a valid state of the game

- p1_turn: checks if both players have the same number of moves for given board
- p2_turn: checks if p2 has one fewer move(s) than p1 for given board

Win Condition Checks: (for computational purposes, we look for 3 in a row to win)

- vertical_w: Checks if a player has won vertically by examining the moves of a given player, and is determined a win if 3 vertically adjacent moves are present
- horizontal_w: Checks if a player has won horizontally in a similar process as vertical, but looks for 3 horizontally adjacent moves
- diagonal_w: Checks if a player has won diagonally by examining the indices of a given player, and is determined a win if 3 diagonally adjacent moves are present
- won: Checks if the given player has won either vertically, horizontally, or diagonally
- finished: Checks if the given board has a player who has won.
- p1_canwin: Checks if some valid move exists for p1 that would allow them to win the game on the given board.
- p2_canwin: Does the same for p2.

Tracing:

- Defines the initial state of the game (generally no moves for either player, but can start in a specific state if desired)
- Varieties of the move transition for some Row r and Column c
    - Checks that no piece already exists at r→c
    - Checks that either r is the bottom row or there is a piece below (r,c) (abides by gravity, this quality is not present in our zero gravity move option)

- p1 gains r→c if it is p1's turn, p2 gains r→c if it is p2's turn
- Player whose turn it is not keeps the same set of moves
- Previous state of the game cannot be finished
    - This means that play does not continue past one player winning
- In moveIntel, players are "intelligent" and will always make a game-winning move if one is available to them

Application:

We were able to correctly model a scaled variety of Connect Four, both with the initial attempt and the final extensible model. The various fixed properties of a C4 game, with specified win conditions and placeable slots, were completed, allowing us to fulfill the foundational goals of our project.

The extensible model allowed for a much more complicated setup, meaning we could vary starting boards to model specific situations, the dimensions of the board, the number of pieces needed for a win (in our model, we are using Connect 3), and the gravity or no gravity condition of the game. We were able to prove the robustness of these conditions as well as the factors that lead to the bounds on Forge's output. We found it too expensive to model a true 6 by 7 board in Forge, and instead worked within the scope of a 4 by 4, Connect 3 board for most of our outcomes, both for speed and readability. Our initial goal was to prove that player one can guarantee a win even if player two plays "perfectly". The conditions for "perfect" play require a number of weighted decisions outlined in our references link below. Because of the limitations of Forge's syntax and the sheer complexity of these calculations, we quickly realized the difficulty of fully expressing a perfectly intelligent player. We instead made the model so that should the full algorithm be delineated in Forge syntax, it could be proven using the game and trace setup. Our attempt at an intelligent player is able to end games if a win is available, pruning all outcomes with nonsense moves, and is tested at the end of the field with our specified starting board. Given a board state from which this player should be able to guarantee a win (in which it is their turn and a winning move is available), we prove that they always will by showing that there is no trace under these conditions in which p2 wins. Given any player algorithm able to be represented in Forge, we are able to prove or disprove its ability to guarantee a win from any possible condition. While we were not able to represent the algorithm of a perfect player and meet our initial goal, we succeeded in establishing a framework capable of proving any algorithm's effectiveness in a variety of game conditions.

References:

http://web.mit.edu/sp.268/www/2010/connectFourSlides.pdf
https://en.wikipedia.org/wiki/Connect_Four