Azul vs. Rojo

Who is Rojo?

When we started the project we searched for previous work done with Azul and artificial intelligence. While reading we saw discussions of Monte Carlo Tree Search, neural networks, and the Rojo AI. The Rojo AI is an implementation of the game in JavaScript with a function that chooses the best move for a player doing board analysis.

We implemented our AI in a really good programming language: Racket. We made our own heuristic function for finding the best move for a player. We use the Rojo AI as a benchmark to test our implementation against. For example, we have run one of our minimax players against three Rojo players.

Minimax*

For our AI, we started with the minimax algorithm we learned in this course. Azul is labeled as a 2-4 player game, so we wanted to have an AI that could win 4-player games. Really, we want it to win *n*-player games, so we extended the game to support an arbitrary amount of players. This lead to having a version of minimax that could work on games with more than 2 players, we refer to it as minimax*.

For the minimax* algorithm, we have to determine a way for one player to compete with the other players. This means having a list of the best score for each player. In order to properly update the best possible value, we have a family of *g* functions. A *g* function determines a player's score based on the other players' scores. The *g* function takes in a sequence of player scores and a player, and computes the difference in the players score and some computation of the competitors' scores. We have a few different computations:

- Sum the other players' values
- Average the other players' values
- Take the max of the other players' values

Each of these is a different *g* function, and we found that the *g* function using average does a bit better than the other ones.

Filtering possible moves

Since a super high branching factor was making minimax useless (143 possible branches on a 1v1 game!), we took inspiration from Rojo AI and tried to greedily pick a few moves from all possible valid moves.

This move-filtering not only makes our implementation more efficient but makes our AI perform better than when we were not filtering any moves.

With move-filtering enabled (keeping just the top 15 moves), and using alpha-beta pruning with a recursion depth of 4, our AI beat Rojo AI around 77.77% of the time (n=100).

A remark on alpha-beta pruning

To do minimax*, we could have tried to generalize alpha-beta pruning like we did with minimax in class. However, in the average case, alpha-beta pruning does not reduce the asymptotic branching factor when used on a game with more than 2 players*.

*(Korf, *Multi-player alpha-beta pruning*, https://www.cc.gatech.edu/~thad/6601-gradAI-fall2015/Korf_Multi-player-Alpha-beta-Pruning.pdf*)*