Udacity AIND
Project 3

Adversarial Game Playing Agent

Synopsis

In this project, you will experiment with adversarial search techniques by building an agent to play knights Isolation. Unlike the examples in the lecture where the players control tokens that move like chess queens, this version of Isolation gives each agent control over a single token that moves in L-shaped movements--like a knight in chess.

Isolation

In the game Isolation, two players each control their own single token and alternate taking turns moving the token from one cell to another on a rectangular grid. Whenever a token occupies a cell, that cell becomes blocked for the remainder of the game. An open cell available for a token to move into is called a "liberty". The first player with no remaining liberties for their token loses the game, and their opponent is declared the winner.

In knights Isolation, tokens can move to any open cell that is 2-rows and 1-column or 2-columns and 1-row away from their current position on the board. On a blank board, this means that tokens have at most eight liberties surrounding their current location. Token movement is blocked at the edges of the board (the board does not wrap around the edges), however, tokens can "jump" blocked or occupied spaces (just like a knight in chess).

Finally, agents have a fixed time limit (150 milliseconds by default) to search for the best move and respond. The search will be automatically cut off after the time limit expires, and the active agent will forfeit the game if it has not chosen a move.

Option 1:

Develop a custom heuristic

must not be one of the heuristics from lectures, and cannot only be a combination of the number of liberties available to each agent

- Create a performance baseline using `run_search.py` (with the `fair_matches` flag enabled) to evaluate the effectiveness of your agent using the #my_moves - #opponent_moves heuristic from lecture
- Use the same process to evaluate the effectiveness of your agent using your own custom heuristic

Experiments:

On my experiments different heuristic models were tried. There were 4 different models tried. These are listed below:

Defensive:
Heuristic defensive model tried to maximize the number of moves.

Aggressive:
Heuristic aggressive model tries to reduce the number of moves of the opponent.

Used calculation: AIMoves - 3 * OpponentMoves

Progressively Aggressive:
This model increases aggressiveness based on the number of the game counts.

Used calculation AIMoves - ply_count * OpponentMoves

Regressively Aggressive:
This model increases aggressiveness based on the number of the game counts. After 5 matches has been played the aggressiveness of the AI would stay as constant 1.

Used calculation AIMoves - Max(1, 5 - ply_count) * OpponentMoves

Halfway Change Heuristic

In this model aggressiveness changes from defensive to more aggressive immediately in the halfway of the game.

2 * AIMoves - OpponentMoves to AIMoves - 2* OpponentMoves

Results

Per heuristic model played 200 times

Start Depth = 1

	E1 (Agg)	E2 (Deff)	E3 (Prog Agg)	E4 (Reg Agg)	E5 (Halfway)
SELF	60 %	50 %	62.5 %	52.5%	42.5%
GREEDY	90 %	95 %	95.0 %	95.0%	97.5%
RANDOM	95 %	97.5 %	97.5%	97.5%	97.5%
MINIMAX	70 %	90%	87.5 %	80.0%	72.5%

Start Depth = 4

	E1 (Agg)	E2 (Deff)	E3 (Prog Agg)	E4 (Reg Agg)	E5 (Halfway)
SELF	57.5 %	37.5%	47.5 %	52.5 %	40 %
GREEDY	85 %	97.5 %	100 %	92.5 %	100 %
RANDOM	85 %	100 %	85 %	95.0 %	100 %
MINIMAX	70 %	82.5 %	65.0 %	77.5 %	90 %

What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during the search?

I try to use offensive and aggressive playing techniques to my opponent.

The experiments based on aggressiveness of the AI.

In the experiments AI gives more space to the opponent first and try the opposite in the second one. On the third and fourth heuristic I slightly change the experiment based on the play count. On the fifth experiment 50 chosen as halfway constant value. move and change the aggressiveness in the middle of the game.

Halfway changing and aggressive heuristic methods were very successful with greedy, random and minimax algorithms. Defensive heuristic model was very successful with self model.

Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?

When i started my depth with 4 instead of 1 it did not change so much performance as time in my tests. Some of the tests win performance sometimes reduced instead of increased. On the third experiment aggressiveness level started much higher at the beginning and continued.

When the time limit changes there was not significant improvement on the results. The time limit set to 300 Depth increased compared to 150 ms. But that didn't affect the results.