Adversarial Game Playing Agent

This is an analysis of an adversarial game playing agent designed to play the game of knights isolation.

Knights 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).

The Project

In this project, 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.

The agent is evaluated by running games against another opponent agent and counting the percentage of wins by the agent.

The opponent agent uses a random choice for moves in the first 2 turns or plies (https://en.wikipedia.org/wiki/Ply_(game_theory)). Beginning with the 3rd ply, the opponent agent uses a Minimax search algorithm to choose the move it will take (https://en.wikipedia.org/wiki/Minimax). The scoring heuristic used by the opponent will be discussed later, as it will be used as a baseline heuristic to compare with a custom heuristic used in the final testing.

The Agent

In order to improve performance over a baseline agent like the opponent agent described above, a number of potential improvements were implemented and evaluated.

An opening book was created similar to those used in other board games such as chess (https://en.wikipedia.org/wiki/Chess_opening_book). The book was created by playing numerous games (over 500,000) and determining which actions led to the best outcomes in each of the states that occurred. The book saved states and actions from the first 4 plies of each game. Preliminary testing showed not much improvement using all 4 plies. In the end, the opening book was only utilized in the first two plies and in the final testing was compared with random choices in the first 2 plies.

The Minimax algorithm was utilized, and was also implemented with Alpha-Beta pruning. In the final testing both were compared.

In both the Minimax search, and Minimax with Alpha-Beta pruning search, iterative deepening was implemented up to a possible depth of 10 plies (https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search). In the final testing the searches with iterative deepening were compared to those without that utilized one search to a depth of 3 plies, similar to the opponent baseline agent.

The Minimax search algorithm uses a heuristic scoring function to evaluate the game state and choose an action from the potential actions available when it hits a maximum depth. A custom heuristic was implemented and evaluated compared with a baseline heuristic. The baseline heuristic was the same as that utilized by the opponent agent, the number of moves (liberties) available to the agent minus the number of moves available to the opponent.

Baseline heuristic:

(number of agent moves) – (number of opponent moves)

The custom heuristic used both of those number of moves, but also utilized information about how far each agent was from the center of the board

Custom heuristic:

*((opponent distance to center + 1) * (number of agent moves)) -*

*((agent distance to center + 1) * (number of opponent moves))*

Final Comparison

The different options discussed above were then tested. The testing ran 1000 games in each instance and the winning percentages for the agent are listed in the table below.

Percentage Of Games Won By Agent

Search Algorithm	Minimax				Minimax with Alpha-Beta Pruning			
Iterative Deepening	No		Yes		No		Yes	
Opening Book	No	Yes	No	Yes	No	Yes	No	Yes
Baseline Heuristic	50.0%	50.0%	61.9%	64.3%	50.0%	50.0%	62.4%	64.7%
Custom Heuristic	53.6%	51.7%	69.7%	70.7%	54.2%	55.7%	69.1%	72.0%

Custom Heuristic Q&A

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

As discussed above, the custom heuristic incorporates two features of each agent, the number of moves/liberties that the agent has, and the distance of the agent from the center of the board. The number of moves the agent has is important because the game ends and the agent loses when that number becomes zero for an agent. The distance to the center was chosen because a location in the center in general will give states with more potential moves in future plies. States far from the center in general have less moves due to the fact that agents cannot move outside the grid.

2. 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?

In testing the agent, both the baseline and custom heuristics would go to the maximum depth with a similar frequency in the tests with iterative deepening. Both heuristics are calculated with a similar subtraction, and the distance to the center of the board was accessed using a lookup table and not calculated each time the score was calculated. For this reason, it would appear that search speed was not the reason for the better performance using the custom heuristic. It is more likely that incorporating the distance from the center added more accuracy.