Project 3: Knights Isolation

<rant>First I just want to point out that the supporting code for this project is confusingly-structured. I know you the reviewer didn't write it, but I spent most of my time: (1) trying to figure out what the `queue` was and how `put`ting to it multiple times was actually the right thing to do (overwrite behavior), (2) extending code I wasn't supposed to have to modify to give me cumulative node and layers searched counts so I could say something intelligent about the behaviors of my heuristics, while also being careful not to break my player's ability to jive with the unmodified files.

(2) was particularly upsetting. The reason `self.context` is the only way to pass information between turns isn't adequately explained and really boils down to not having a good way to truncate the execution time of a function call without starting and killing a separate thread or process (https://stackoverflow.com/questions/14920384/stop-code-after-time-period), true parallelism in Python being only processes and not threads because of the Global Interpreter Lock, the difference between processes and threads, and the way information has to be carefully piped between processes. Much of that is such language-specific minutiae. I understand we weren't meant to need to grok it, but it becomes relevant. Then the players are not even passed up to the main caller, so I had to go in and add them to the `return` statement of `_play` and handle them myself through the rest of the chained method calls.

This code is obviously functional, but when it takes an experienced software engineer several hours to trace it all out, it's poorly written/documented. And when getting cumulative nodes-expanded counts is so non-trivial, why suggest/require we do it in the instructions without actually explaining how the code is structured? This detracts from the substance of the algorithms and the point of the project. That's energy I now don't have to put toward an opening book or other improvements.</rant>

I'm choosing **Experiment 1, Custom Heuristic**.

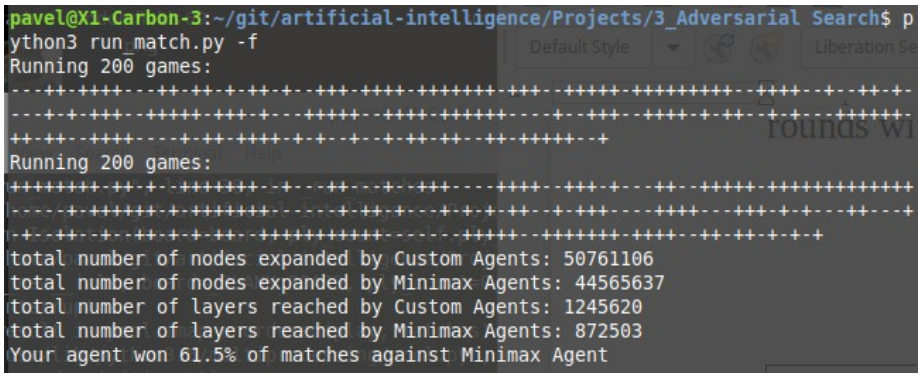
First, if you didn't read the rant, you should know I modified the project code to return players up to the caller, and players keep track of cumulative nodes-visited and layers-reached counts. These should be helpful in comparing heuristics to each other, beyond just how many times they beat a baseline opponent.

Technical Notes:

- The number of nodes is simply the number of times `min_value`, `max_value`, and the top-level search function (either `alpha_beta_search` or `minimax`) get called. And the number of layers is the sum of the `depth`s passed to that search function across all calls in my iterative deepening loop.
- I had a bug where at the end of the game the player would quickly explore the rest of the tree all the way down to the leaves, then do the same search again and again and again with `depth` getting up to the hundreds or thousands really quickly, causing the cumulative layers explored to *exceed* the number of nodes explored. How can I keep track of when I've reached the leaves and stop? We can expect the number of layers left is about $\log_{\text{branching\_factor}}$(number of nodes left), but in the most extreme case the last few moves leave only one option available to a player, so the tree looks more like a list, and the number of layers left = the number of nodes left. So in my code, if the `depth` equals the number of nodes explored by the search call, I know that's the maximum depth the search could actually be going, and the function returns, does no further iterative deepening. This makes my counts much more intuitively reasonable.

For my baseline, I'm using Minimax with iterative deepening, which is just a tiny improvement on the Minimax agent provided, to take full advantage of all the compute time. It didn't seem fair or interesting to use a Minimax that could only expand *way* fewer nodes.

When I run, it ends up looking something like this:



And here is a table with more comprehensive results:

NUM_ ROUNDS	TIME_ LIMIT	Player 1	Player 2	Player 1 Win %	Player 1 cumulative nodes visited	Player 2 cumulative nodes visited	Player 1 cumulative layers reached	Player 2 cumulative layers reached
100 (fair)	150ms	Minimax w/iterative deepening	Minimax w/iterative deepening	50.5%	51,822,571	51,913,438	1,182,007	1,198,220
100 (fair)	150ms	Alpha-beta w/ #liberties - #opponent liberties heuristic	Minimax w/iterative deepening	61.5%	50,761,106	44,565,637	1,245,620	872,503
100 (fair)	300ms	Alpha-beta w/ #liberties - #opponent liberties heuristic	Minimax w/iterative deepening	58.8%	42,986,749	37,231,211	1,126,759	726,329
100 (fair)	2000ms	Alpha-beta w/ #liberties - #opponent liberties heuristic	Minimax w/iterative deepening	59.8%	292,113,769	245,895,977	4,788,738	3,255,994
100 (fair)	150ms	Alpha-beta w/ chase opponent heuristic	Minimax w/iterative deepening	49.0%	31,612,990	22,051,068	722,181	435,385
100 (fair)	150ms	Alpha-beta w/ avoid opponent heuristic	Minimax w/iterative deepening	34.5%	25,297,201	17,220,627	55,1392	33,0988
100 (fair)	150ms	Alpha-beta w/ favor center heuristic	Minimax w/iterative deepening	51.8%	26,633,780	17,861,917	64,0803	39,5877
100 (fair)	150ms	Alpha-beta w/ weighted sum of center and liberties heuristics	Minimax w/iterative deepening	61.8%	22,167,881	20,391,727	736,658	511,124
100 (fair)	150ms	Alpha-beta w/ liberties of liberties heuristic	Minimax w/iterative deepening	75.5%	14,470,647	22,175,592	728,533	554,522

Questions

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

My heuristics incorporate player positions. This is mostly based on my knowledge of chess.

Knights can move roughly from the center to edge of a circle, and they have difficulty controlling spaces too near to themselves. So is it possible to stay inside that ring, and thereby keep your opponent from being able to block your liberties?

The center of the board is considered stronger than the edges, especially for constant-distance movers like knights. I.e. pieces typically control more territory when there. So ending with your opponent on the edge while in the center yourself is more likely to leave you with liberties.

And finally, I've added a heuristic that's a weighted combination of my position-based heuristics and the liberties-based one, so more expensive to compute but hopefully more accurate.

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?

It's clear the alpha-beta agent with the liberties-based heuristic can look deeper (more cumulative layers) and examine more nodes than the minimax agent. This makes sense, because it's pruning branches and using fast evaluations at really deep levels. What's surprising to me is that even with much longer thinking times the win percentage doesn't increase. This indicates the search space is enormous enough that looking ~50% deeper doesn't actually "solve" the game all the way down to the leaves, just makes your decisions more informed, where that information is still mostly based on heuristic guesses, which can be noisy and so translate to only about 10% more actual wins.

Probably for that reason, better heuristic guesses are really important here. Most of my distance-based heuristics don't turn out to be super helpful, but the combined liberties-and-center-distance-based heuristic does about as well as the liberties heuristic alone while examining fewer nodes and layers in equal time. So clearly that distance method carries some useful information, but the benefit is canceled out by the extra evaluation time preventing the search from reaching as deep. However, a heuristic based on liberties of liberties does remarkably better despite the extra evaluation time and fewer nodes explored. This is a case of the tradeoff being worthwhile, because the heuristic is just that much more accurate, because liberties are really what matter in this game.