For this Isolation playing agent project, I tried the following three heuristic function ideas, which all build on top of the "improved score" function (# of player moves - # of opponent moves): 1) "guaranteed moves" function that only counts number of moves guaranteed for the active player, 2) "percent score" that looks at the # of moves as percent of number of all blank spaces, and 3) "single branch depth search" function that looks at the next step if there's only one possible move available. Options 1 & 3 were strategies I used, and option 2 is something I just tried that had an effect of a strategy I used.

Idea behind "guaranteed moves" is that you cannot make a move based on a following move that can be blocked. If there are *n* available moves for the inactive player, the number of guaranteed moves is *n-1* if at least one of those moves overlaps with the active player's available moves.

Idea behind "percent score" should be obvious. Its effect, however, is not. For a given board state, let's say # of blank spaces is *N*. For inactive player, he will have *N-1* blank spaces, because the active player will have taken another space! Since dividing by *N-1* is bigger, it slightly prefers # of inactive player moves compared to # of active player moves. In considering a next move, this has the effect of emphasizing increasing # of my moves vs reducing # of opponent moves.

Idea behind "single branch depth search" is that if there's only one available move, then it should play it all the way out, either until there are more moves for a score or the game ends in a win or a loss. This quickly either decides on a winning path or avoids on a losing path.

Below is a table summarizing winning percentages of the four agents (ID_Improved, Guaranteed moves, Percent score, and Single branch depth search) against the 7 test

	Random	MM_Null	MM_Open	MM_Improved	AB_Null	AB_Open	AB_Improved	**Subtotal**	ID_Improved	Guaranteed moves	Percent score	Single branch depth search
ID_Improved	89.7	84.0	70.0	67.0	75.0	66.0	61.3	**73.3**	50.3	49.5	46.0	44.0
Guaranteed moves	92.3	84.3	69.3	70.7	76.0	70.7	62.7	**75.1**	50.5	50.5	50.8	51.0
Percent score	93.0	82.3	73.3	70.0	76.0	68.7	68.0	**75.9**	54.0	49.2	50.3	44.5
Single branch depth search	92.3	84.3	72.3	71.0	77.3	67.3	66.3	**75.9**	56.0	49.0	55.5	49.5

Table 1. Table of winning percentages between different agents. The results against the 7 test agents are from 300 games each, while results against each other are from 400 games each. Timeout was left as default, 150 ms.

agents and against each other, even against itself. The winning rates for agents playing against itself are 50±0.5%, telling us there's some uncertainty in these values. Fortunately, the three heuristic functions all performed better than the benchmark ID_Improved, both their performance against test agents and against each other. This may be because all three heuristic functions are built on top of the ID_Improved agent. This is not to say more complex model always does better, as it usually slows down the computation.

There's no clear winner though. It appears that against the test agents, "Percent score" and "Single branch depth search" perform the best. However, when played against each other, "Guaranteed moves" beat almost all of them ("almost", because winning rates are relatively close to 50%).

In my opinion, the best algorithm here is the "Single branch depth search" as this algorithm has best performance against test agents, performs well against ID_Improved and "Percent score" (its loss to "Guranteed moves" is by very small margin that may be within uncertainty), and is closest to the strategy I use! One last comment about "single branch depth search" performance is that it also seems to perform little more consistently. Even though performance against each other may be more unpredictable, their performance against the test agents should follow a trend---score against "Improved" should be less than against "Open", which should be less than against "Null", and against "AB" should be less than against "MM". Better agent may be made when some of these are combined, and I'm sure better heuristic functions are out there!

Adrian Yi