Heuristics

I implemented the following heuristics.

cover_opponent_moves

- For a player, how many moves can he make from his current position that directly inhibit the opponent too. i.e. the number of positions that both the player and the opponent are able to reach in one move from their current positions
- As compared to the *improved* heuristic (number of moves of a player - number of moves of the opponent) , this heuristic tries to account for how well a player *controls* the board and forces the opponent's moves and thus captures the notion of *positional advantage*.

look_ahead_by_one

- This heuristic implements a *one move look ahead* strategy. For all the positions reachable from the current position, it computes how good all the positions are and sums up their scores and uses the sum as the score of the current position. This heurisitic thus tries to account for how good a position is by computing how good all its successor positions are in aggregate
- One tweak to this heuristic is that when looking ahead, we ignore *losing* board positions. i.e. if a player has 4 moves available to him and one of them is a position where he loses, then we ignore that position. This seems to help in practice since the scores obtained by this heuristic are better with this tweak. Intuitively, if we have a losing move in the future we expect our agent to take one of the remaining moves so we do not want to have the losing move affect our score.

select_one_of

- This heuristic works as follows. Initially it uses the *improved* heuristic (number of moves of a player - number of moves of the opponent). During the *mid-game* phase, as determined by the number of free squares, it uses the *cover_opponent_moves* heuristic and finally during *end-game* phase, it uses the *look_ahead_by_one* heuristic. The idea of *positional* advantage captured in the *cover_opponent_moves* heuristic is not very useful in the beginning of the game as there are lots of empty squares everywhere. However once the board starts filling up, then, positional advantage becomes more meaningful since it is feasible to try and begin to box in the opponent. And finally, towards the *end-game*, it becomes critically important to not end up in a bad position since there are very few options to move to during this phase of the game. Hence during the

end-game phase it becomes more useful to look ahead and ensure that you are not jumping into a fatal position.

aggregate_heuristics

- This is the one I selected as my *custom* heuristic that performs best. It is a combination of *improved*, *cover_opponent_moves* and *look_ahead_by_one*. It evaluates all three and sums them to obtain a final score.

Here are three runs of the *tournament* script. I have deliberately included one run where *Improved* beats *Custom*

AB_Custom = aggregate_heuristics

AB_Custom_2 = cover_opponent_moves

AB_Custom_3 = look_ahead_by_one

AB_Custom_4 = select_one_of

Opponent	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3	AB_Custom_4
	Won/Lost	Won/Lost	Won/Lost	Won/Lost	Won/Lost
Random	8/2	8/2	8/2	7/3	5/5
MM_Open	5/5	8/2	5/5	6/4	6/4
MM_Center	6/4	8/2	8/2	6/4	7/3
MM_Improved	5/5	7/3	4/6	6/4	7/3
AB_Open	6/4	6/4	5/5	4/6	4/6
AB_Center	7/3	6/4	6/4	5/5	4/6
AB_Improved	4/6	6/4	3/7	6/4	6/4
Win Rate:	58.6%	70.0%	55.7%	57.1%	55.7%

Opponent	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3	AB_Custom_4
	Won/Lost	Won/Lost	Won/Lost	Won/Lost	Won/Lost
Random	8/2	8/2	8/2	7/3	8/2
MM_Open	5/5	7/3	5/5	6/4	8/2
MM_Center	6/4	9/1	7/3	7/3	6/4
MM_Improved	7/3	8/2	5/5	6/4	5/5
AB_Open	4/6	6/4	5/5	5/5	4/6
AB_Center	7/3	7/3	7/3	5/5	4/6
AB_Improved	6/4	6/4	5/5	4/6	4/6
Win Rate:	61.4%	72.9%	60.0%	57.1%	55.7%

Opponent	AB_Improved	AB_Custom	AB_Custom_2	AB_Custom_3	AB_Custom_4
	Won/Lost	Won/Lost	Won/Lost	Won/Lost	Won/Lost
Random	8/2	8/2	8/2	8/2	6/4
MM_Open	6/4	6/4	6/4	5/5	5/5
MM_Center	8/2	8/2	6/4	9/1	9/1
MM_Improved	6/4	5/5	6/4	6/4	4/6
AB_Open	7/3	5/5	6/4	5/5	4/6
AB_Center	6/4	7/3	7/3	5/5	3/7
AB_Improved	5/5	6/4	3/7	5/5	2/8
Win Rate:	65.7%	64.3%	60.0%	61.4%	47.1%

Analysis of the heuristics

In an adversarial search based strategy, there is a fundamental tradeoff between how deep down the tree you can evaluate vs the complexity of the evaluation function applied to the position, since we (usually) have time constraints and need to come up with the next move in a timely manner. So, the time complexity of each of the heuristic is important and is described next.

Amongst the heuristics we considered, *Improved* and *cover_opponent_moves* both are quite simple to implement and are roughly equivalent in terms of time taken to compute them. *look_ahead_by_one* can be more expensive since it evaluates *Improved* for all the positions that the player can move to. *select_one_of* is roughly the same as either *Improved* or *cover_opponent_moves* since it calls one of them for most of the game and only in the end game phase, does it call *look_ahead_by_one*. *Custom* i.e. *aggregate_heuristics* is more expensive than the others since it calls all the other (*Improved*, *cover_opponent_moves* and *look_ahead_by_one*) heuristics and aggregates their scores.

The results of several tournament rounds show two interesting observations about these heurisics:

- Even though *Custom* usually wins, it does not always do so. Thus a more computationally expensive and/or *domain-informed* heuristic does not always win. This drives home the fact that we have a heuristic based approach and not an approach that is mathematically (or otherwise) guaranteed to be better.
- All the heuristics except *Improved* are *domain-aware*. Inspite of that they are not necessarily superior.

Recommendation

So now the question is - which heuristic should be used.

aggregate_heuristics usually wins over the other heuristics but it's drawback is that it is computationally more expensive. Inspite of that, I recommend it because of the following reasons.

- Most h/w today has concurrency of some kind (multiple cores/fpgas etc). So it is possible to do independt work in parallel and so the individual heuristics that comprise *aggregate_heuristics* can be computed in parallel. Even if we ignore concurrency, we can see that there is a lot of overlap in the work done by each of the individual heuristics. So we can in theory write a more optimal single function that does the work of all the individual heuristics and does the overlapping work just once instead of calling the individual heuristics as standalone functions like we do. So in summary, even though this heuristic is computationally more expensive, that is not a show-stopper
- Another interesting observation after running several tournament rounds is that the *aggregate_heuristics* agent very rarely loses against any opponent. An agent using say the *Improved* heuristic may have a higher over all win rate but it will sometimes lose to an opponent (usually *AB_Open* or *AB_Center*. i.e. in the 10 matches it plays against one of these agents, it loses more matches than it wins). Whereas, the agent using *aggregate_heuristics* very rarely loses against any one opponent even if it may end up with a lower over all win-rate. This thus indicates that the *aggregate_heuristics* performs better against a variety of other agents
- The overall structure of *aggregate_heuristics* is that it is a linear composition of other heuristics where each component heuristic has equal weight. This kind of feature composition is a very general technique in many AI algorithms and is known to work quite well in practice. Thus *aggregate_heuristics* can be further refined by adding new features, adding weights to the features and so on, i.e. it has scope for further improvements and tuning