Isolation Heuristic Analysis

Daniel Meechan

Udacity Artificial Intelligence Nanodegree

Term 1, Project 2 (October 2017)

Introduction

As part of the Adversarial Search Agent project, I implemented a minimax algorithm with alpha-beta pruning and iterative deepening to create an AI capable of competing at the board game Isolation. Minimax evaluates possible future game states and attempts to maximise the player's score while assuming that the opponent is trying to minimise the player's score. Alpha-beta pruning optimises the minimax process by eliminating nodes in the search tree which won't affect the final decision. Iterative deepening ensures the algorithm will have a result ready when the move timer expires, while still going as deep as possible by searching at increasingly further depths.

With branching factor *b* and search depth *d*, minimax with alpha-beta pruning performs just $2b^{d/2}$ evaluations (in an optimal situation), which is far fewer than the b^d evaluations performed with minimax.

I created three heuristics for scoring a board state. The scoring heuristics help the AI evaluate different moves.

Heuristic Descriptions

All three heuristics calculate how many legal moves the AI and opponent have available, applies weights to those values, and then subtract the AI's value from the opponent's value.

AB_Custom weights the opponent's moves much more by doing:

$$\textit{my\_moves} - \textit{opponent's\_moves}^{1.5}$$

Using exponents leads the AI to be much more avoidant of nodes where the opponent will have many moves available.

AB_Custom_2 weights both players' moves exponentially, while giving the opponent's value more weighting:

$$\textit{my\_moves}^{2} - \textit{opponent's\_move}^{2.5}$$

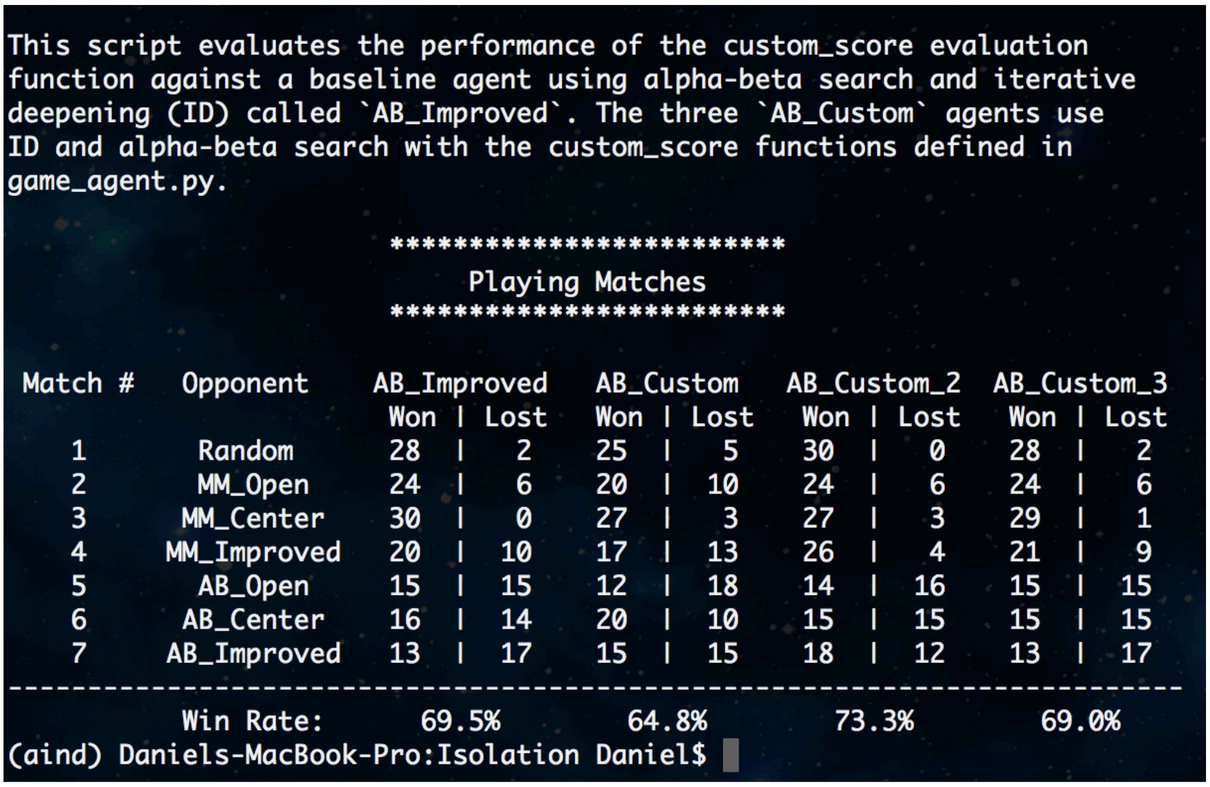
The idea is that the AI will value nodes much more if they give it more moves, but will be incredibly wary of nodes which gives the opponent many moves.

AB_Custom_3 is the most aggressive heuristic because it gives very little weighting to the AI's moves, while giving a lot of weighting to the opponent's moves:

$$\textit{my\_moves}^{1/2} - \textit{opponent's\_move}^{2}$$

Results and Recommendation

Heuristic AB_Custom_2 scored better than the other two heuristics, with AB_Custom scoring worst (see figure 1).



```
This script evaluates the performance of the custom_score evaluation
function against a baseline agent using alpha-beta search and iterative
deepening (ID) called `AB_Improved`. The three `AB_Custom` agents use
ID and alpha-beta search with the custom_score functions defined in
game_agent.py.

                        *************************
                             Playing Matches
                        *************************

 Match #   Opponent    AB_Improved   AB_Custom   AB_Custom_2  AB_Custom_3
                        Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      28  |   2    25  |   5    30  |   0    28  |   2
    2       MM_Open     24  |   6    20  |  10    24  |   6    24  |   6
    3      MM_Center    30  |   0    27  |   3    27  |   3    29  |   1
    4     MM_Improved   20  |  10    17  |  13    26  |   4    21  |   9
    5       AB_Open     15  |  15    12  |  18    14  |  16    15  |  15
    6      AB_Center    16  |  14    20  |  10    15  |  15    15  |  15
    7     AB_Improved   13  |  17    15  |  15    18  |  12    13  |  17
--------------------------------------------------------------------------
           Win Rate:      69.5%        64.8%        73.3%        69.0%
(aind) Daniels-MacBook-Pro:Isolation Daniel$
```

Figure 1: Tournament.py results for 30 matches with a time limit of 150ms

I recommend using AB_Custom_2 because it achieved the greatest win rate and scored higher than the AB_Impoved heuristic (which scored 69.5%). This is likely due to the weightings giving nodes with more moves exponentially higher values, while giving slightly more weighting to the opponent’s moves.

Heuristic	AB_Custom	AB_Custom_2	AB_Custom_3
Win Rate	64.8%	73.3%	69.0%