Isolation Game: Heuristic Analysis

Artificial Intelligente Nanodegree

Celso Araujo

November 13th, 2017

Introduction

This report is a requirement of the “Build a Game-Playing Agent” project included in Udacity's Artificial Intelligente Nanodegree. The project description, this report and all code, including code for my solution, can be found on my GitHub repository: https://github.com/celsofaf/AIND-Isolation.

The aim of this report is, according to the project instructions:

- To describe the three heuristics I developed for calculating board-state scores for the Isolation game. These scoring heuristics are used to build 3 playing agents (called simply “players” here) ;
- Show and comment the results of the “Tournament” comparing these 3 players to other 7 pre-implemented players, including a specific one which should be considered as a benchmark.

The 7 pre-implemented players

Player #1 – Random: as its name suggests, simply choses a random move among the available ones, and plays this move.

Players #2 to #7 are the cartesian product of two gameplay heuristics and three board-state score heuristics. The gameplay heuristics are:

- **MiniMax – MM:** chooses a move using depth-limited minimax search over the game tree
- **AlphaBeta – AB:** chooses a move using iterative deepening minimax search with alpha-beta pruning over the game tree

And the score heuristics are:

- **Open:** score is equal to the number of possible moves available to the player
- **Center:** score is equal to the squared euclidean distance of the player's current position to the board center
- **Improved:** score is equal to the number of possible moves available to the current player (Open score) minus the number of possible moves available to the opponent player.

So, the remaining pre-implemented players are:

- **Player #2 – MM_Open:** MiniMax gameplay with Open score heuristics
- **Player #3 – MM_Center:** MiniMax gameplay with Center score heuristics
- **Player #4 – MM_Improved:** MiniMax gameplay with Improved score heuristics
- **Player #5 – AB_Open:** AlphaBeta gameplay with Open score heuristics

- **Player #6 – AB_Center:** AlphaBeta gameplay with Center score heuristics
- **Player #7 – AB_Improved:** AlphaBeta gameplay with Improved score heuristics

Note: a MiniMax player here will always search the game tree to a maximum depth of 3, while an AlphaBeta player will use iterative deepening technique to search the tree starting at depth = 1 and going as deep as the game timer allows.

The benchmark AI player is **Player #7 – AB_Improved**. My aim is to beat this player under the project constraints.

The 3 new players

The new players use alpha-beta prunning with iterative deepening (like the "AB" pre-implemented players) to choose the next move, but with score functions that were implemented by myself. These are the players with their score functions:

- **AB_Custom:** score = (number of player possible moves) – 3*(number of opponent possible moves). This is a modified version of Improved score, possibly aiming for a more defensive style of playing.
- **AB_Custom_2:** score = 3*(number of player possible moves) – (number of opponent possible moves). This is a modified version of Improved score, possibly aiming for a more aggressive or offensive style of playing.
- **AB_Custom_3:** score = (number of player possible moves) + (number of player possible second moves) – (number of opponent possible moves). Here, I "allowed" the player to make two moves in a row so as to look a little ahead into the game, counting the possible unique "first" and "second" moves. The more moves a player is able make in two turns, the better. From this number, I subtracted the current number of possible moves for the opponent, because I understand its "freedom" is also to be accounted for. ;-)

AB_Custom, AB_Custom_2 and AB_Custom_3 players will be collectively referred to as "my players" from now on.

Measuring players performances

Two "tournaments" were set up in order to assess the players performances. Players were divided in two groups, and member of one group played against all members of the other group. The groups were:

- AB_Improved, AB_Custom, AB_Custom_2 and AB_Custom_3 (the benchmark player plus my players)
- Random, MM_Open, MM_Center, MM_Improved, AB_Open, AB_Center and AB_Improved (all 7 pre-implemented players)

Note that the benchmark player, AB_Improved, will end up playing against itself.

So, a tournament will have 28 (4 x 7) sets of matches. Each match will, in fact, be a double match: first round with random starting positions, and second round with the same random starting positions, but with players reversed as to whom starts.

Tournament #1 consists of 200 matches between each pair of players (that is, 400 rounds, with 400 x 28 = 11,200 rounds played in total), with a time limit of 150 milliseconds for each move. Tournament #2 has the same number of matches, but with an increased time limit: 450 milliseconds for each move. This is made to allow the search heuristics to go deeper in the game trees and see the effects.

Results

Tournament #1 took 9h28min to run on my computer. Here are its results:

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	382	18	380	20	370	30	383	17
2	MM_Open	296	104	294	106	306	94	299	101
3	MM_Center	361	39	345	55	363	37	345	55
4	MM_Improved	282	118	265	135	299	101	290	110
5	AB_Open	216	184	219	181	207	193	216	184
6	AB_Center	230	170	232	168	232	168	247	153
7	AB_Improved	210	190	208	192	192	208	209	191
	Win rate:	70.6%		69.4%		70.3%		71.0%	

Tournament #2 took 28h42min to run on my computer. Here are its results:

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	374	26	369	31	362	38	372	28
2	MM_Open	301	99	312	88	304	96	296	104
3	MM_Center	351	49	356	44	337	63	337	63
4	MM_Improved	289	111	282	118	280	120	304	96
5	AB_Open	200	200	196	204	201	199	221	179
6	AB_Center	209	191	212	188	211	189	223	177
7	AB_Improved	199	201	205	195	203	197	196	204
	Win rate:	68.7%		69.0%		67.8%		69.6%	

Comments and conclusion

- On average, on both tournaments, AB_Custom_3 player had the best win rate.
- My players are consistently better than any MM pre-implemented player. This is something to be expected, as (1) my players are AB players, and AB is expected to search deeper than MM under the same time constraints, and (2) MM is constrained to depth = 3, where AB uses iterative deepening with no depth limit.
- On Tournament #1, my players consistently beat AB_Open and AB_Center. However, AB_Open is a more difficult opponent in Tournament #2 – which allows for 3x more “thinking time”. A possible explanation is: AB_Open's score function is very quick to be evaluated, so it is probably able to search deeper in the game tree, thus being more able to find a winning move.
- The same explanation as above can explain why AB_Custom_3 won over AB_Improved on Tournament #1 but lost on Tournament #2. AB_Custom_3 uses a more sophisticated score function, which can be a better one better when it's constrained to a smaller number of

function evaluations, but this advantage drops when the opponent is able to search even deeper into the tree because of its simpler, yet meaningful, score function.

- Which one of my players would I choose, then? AB_Custom_3 is the safest bet.