Heuristic Analysis - Build an Adversarial Game Playing Agent

By Dharmanandana Reddy

Introduction

build adversarial game playing agent is to play isolation game by utilizing two types of search agents are Minimax and Minimax search with Alpha-Beta pruning and iterative deepening to improve the performance of Minimax search

Minimax/Alpha-Beta uses the heuristic i.e., count the number of possible custom agent moves subtracted by the number of possible opponents moves.

It was observed that Minimax with alpha-beta pruning and iterative deepening performed better than vanilla Minimax search.

As we know agents goal is to win the game, so evaluation function is required which can give the score of players in each game state according to situation. Players move in L-shaped from current position, players with no legal move, which means lose the game.

Evaluation functions:

Available moves difference: it means that the number of the possible moves the custom agent can perform subtracted by the number of possible moves the opponent can perform n-ply ahead, which returns value considered as game score, this function returns higher value when the player has more legal moves than opponents

Ratio of available moves: gets number of legal moves of each player, divides legal moves of active player with legal moves of inactive player, which returns value considered as a game score

Baseline heuristic: #my_moves - #opponent_moves

Custom heuristic: #my_moves-2*#opponent_moves+#my_future_moves-2*#opponent_future_moves

Following commands was run to fetch the results

```
root@8aa6ff2215a1:/home/workspace# python run_match.py -f -r 50 -o MINIMAX -p 3 -t 300
```

```
root@8aa6ff2215a1:/home/workspace# python run_match.py -f -r 50 -o GREEDY -p 3 -t 150
```

Running 100 games:

```
+-+++-+++++++++++++++++++--+++++++-+++-++-+++-+++++++++++++-+++-+-++-----+-+++---
+++++++++++++++++++-

Running 100 games:

++-+++++-++++++++-+++++++++-++++-++++++-+++++++--+-++--++++++++++++++++-++++-
++++++++++++++++++-+-++-

Your agent won 81.5% of matches against Greedy Agent
root@8aa6ff2215a1:/home/workspace# python run_match.py -f -r 50 -o GREEDY -p 2 -t 150

Running 100 games:

+-+-+-+++++++-+++++-++++++++++--+++++++++++--++-++-+++++++++++++-++--
+++++++++++++++++++++++++++++-+-+

Running 100 games:

+-+-+++++++++-+-++-+++++++++---+++++++++--++++-+++-++++-
++++++++++++++++++++++-+++++++++-+++++++++++

Your agent won 84.5% of matches against Greedy Agent

root@8aa6ff2215a1:/home/workspace#
```

Performance Results:

Opponent	#Matches	Time-limit	Player-win %	No of processes employed
Minimax	100	150	65.5	2
Minimax	100	150	67	3
Minimax	100	300	61	3
Greedy	100	150	81.5	3
Greedy	100	150	84.5	2

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

The custom player considered current liberties and subsequent liberties, multiplies the opponent current and future liberties by factor of 2, this is done to penalize the player to pick stronger moves which helps the player to gain maximum number of open liberties and better chance of winning the game

- 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?

The Greedy agent heuristic worked better within 150ms with 84.5% winning compare to Minimax agent, it means that Minimax agent with alpha beta search with iterative deepening works better than greedy agent, so winning chances are high and also search becomes ineffective at deeper levels though time increases.