AIND Project 2: Building a Game-Playing Agent

by Dmitri Lihhatsov / GitHub: @dlihhats

Introduction

The goal of this project is to create an AI agent for a game of Isolation. In our version of the game, players can only move to cells that lie two by one cells away from the player's current location – just like a knight figure in the game of chess. The last player who cannot move any further looses the game.

Our AI agent tries to find an optimal move by traversing down the tree of all possible moves using a Minimax search algorithm with Alpha-Beta pruning and iterative deepening given a time constraint for every move. Our goal is to come up with an heuristic function describing a state of the game that would outperform another predefined AI agent. This predefined AI agent uses the same search method but with a heuristic function equal to the difference between the number of possible player moves and opponent moves.

Evaluation Method

We will evaluate the performance of three heuristic functions `f(game)` in comparison with the predefined AI agent by making them all play 100 games against the following AI opponents:

1. **`AB_Improved`**: Minimax, Alpha-Beta, iterative deepening with `f(game) = my_moves – opponents_moves`
2. **`AB_Center`**: Minimax, Alpha-Beta, iterative deepting with heuristic function evaluating the distance to the centre of the board.
3. **`AB_Open`**: Minimax, Alpha-Beta, iterative deepening with `f(game) = my_moves`
4. **`MM_Improved`**: Minimax with search depth equal to 3 and `f(game) = my_moves – opponents_moves`
5. **`MM_Center`**: Minimax with search depth equal to 3 and heuristic function evaluating the distance to the centre of the board.
6. **`MM_Open`**: Minimax with search depth equal to 3 and `f(game) = my_moves`
7. **`Random`**: An agent that selects its next move randomly.

Heuristic Functions

In order to come up with an heuristic function that would be superior than a simple difference between number of player's moves and opponent's moves, we have to:

- keep the heuristic as simple as possible to allow the algorithm search deeper in the allotted time
- improve the pruning of the tree by highlighting the difference between various game states and finding equivalent game states.

Thus, we evaluated the following heuristic functions.

1. **`AB_Custom`**: This agent uses a function that capitalises on the successful difference between player's moves and opponent's moves but also considers the ratio. We reckoned it to be more accurate than simply calculating the difference because it distinguishes various game states that the `AB_Improved` agent finds equivalent. For example, considering two game states, 1) when the player has 5 moves and the opponent has 4 moves; and 2) when the player has 8 moves and the opponent has 7 moves; `AB_Custom` agent will prefer 1) while `AB_Improved` considers them to be equivalent.

   ```
   f = own_moves – opp_moves + own_moves / opp_moves
   ```

2. **`AB_Custom_2`**: This agent uses a function that evaluates the ratio between the number of player's moves and opponent's moves. We wanted to compare its performance with a slightly more complex heuristic function of `AB_Custom`.

   ```
   f(game) = own_moves / opp_moves
   ```

3. **`AB_Custom_3`**: With this agent we decided to try out a function that would concentrate at the top left part quadrant of the board. We wanted to check an assumption that focusing on a particular part of the board could bring and advantage. We didn't have much hope for this function but as we will see, it still performed quite well.

   ```
   f = own_moves_in_top_left - opp_moves_in_top_left
   ```

Evaluating Performance

We run 100 games between all opponents with each opponent making the first move in 50 games and randomly determining the initial location of the players. These games are independent, and we will run a Z-test to determine if the difference between win rates of different agents is statistically significant.

	X_1	X_2
1	a	b
0	c	d
$\sum$	n_1	n_2

$$\hat{p}_1 = \frac{a}{n_1}$$

$$\hat{p}_2 = \frac{b}{n_2}$$

$$\text{Confidence interval for } p_1 - p_2\text{:}\quad \hat{p}_1 - \hat{p}_2 \pm z_{1-\frac{\alpha}{2}} \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}$$

$$Z - test : Z(X_1, X_2) = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{P(1-P)(\frac{1}{n_1} + \frac{1}{n_2})}}$$

$$P = \frac{\hat{p}_1 n_1 + \hat{p}_2 n_2}{n_1 + n_2}$$

Results

```
                                *************************

                                     Playing Matches

                                *************************


 Match #    Opponent    AB_Improved    AB_Custom    AB_Custom_2  AB_Custom_3
                         Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1      AB_Improved   48  |  52    55  |  45    56  |  44    52  |  48
    2       AB_Center    57  |  43    55  |  45    58  |  42    52  |  48
    3        AB_Open     47  |  53    54  |  46    54  |  46    48  |  52
    4      MM_Improved   87  |  13    81  |  19    89  |  11    80  |  20
    5       MM_Center    91  |   9    95  |   5    93  |   7    92  |   8
    6        MM_Open     88  |  12    88  |  12    92  |   8    88  |  12
    7        Random      98  |   2    97  |   3    99  |   1    100 |   0
--------------------------------------------------------------------------
           Win Rate:      73.7%        75.0%        77.3%        73.1%
```

```
Z-test

AB_Improved vs. AB_Custom

95% confidence interval for a win probability, AB_Improved: [(0.703, 0.768)]

95% confidence interval for a win probability, AB_Custom: [(0.717, 0.781)]

95% confidence interval for the difference: [(-0.059, 0.033)]

p-value, one-sided, AB_Custom is better than AB_Improved: 0.291

p-value, two-sided, AB_Custom is different from AB_Improved: 0.582


AB_Improved vs. AB_Custom_2

95% confidence interval for a win probability, AB_Improved: [(0.703, 0.768)]

95% confidence interval for a win probability, AB_Custom_2: [(0.740, 0.802)]

95% confidence interval for the difference: [(-0.081, 0.009)]

p-value, one-sided, AB_Custom_2 is better than AB_Improved: 0.060

p-value, two-sided, AB_Custom_2 is different from AB_Improved: 0.120
```

Conclusion

Z-test shows that unfortunately with the level of confidence at 95% none of our heuristic functions are statistically any better than the `AB_Improved` agent. However, with a confidence level at 90% we see that the `AB_Custom_2` agent with an heuristic function `f(game) = own_moves / opp_moves` performs better than `AI_Improved` (p-value = 0.060).

Therefore, for an intelligent agent in our game of Isolation, we recommend using a minimax algorithm with alpha-beta pruning, iterative deepening and an heuristic function equal to the ratio between player’s moves and opponent’s moves for the following reasons:

1. This heuristic function is simple enough to calculate under time restrictions to reach significant depths in the search tree.
2. This heuristic function amplifies the difference between various game states better than a function that calculates the difference between the number of moves. Thus, it allows to select the next move more intelligently.
3. Variability in the scores of different game states provides efficient alpha-beta pruning of the search tree allowing even deeper searches.