Heuristic Analysis

Building a Game-Playing Agent

Submitted By:-

Nishant Mittal

Artificial Intelligence Nanodegree, Udacity

Synopsis

This project aims at developing adversarial search agent to play the game "Isolation".

Isolation is a deterministic, two-player game of perfect information in which the players alternate turns moving a single piece from one cell to another on a board. Whenever either player occupies a cell, that cell becomes blocked for the remainder of the game. The first player with no remaining legal moves loses, and the opponent is declared the winner.

This project uses a version of Isolation where each agent is restricted to L-shaped movements (like a knight in chess) on a rectangular grid (like a chess or checkerboard). The agents can move to any open cell on the board that is 2-rows and 1-column or 2-columns and 1-row away from their current position on the board. Movements are blocked at the edges of the board (the board does not wrap around), however, the player can "jump" blocked or occupied spaces (just like a knight in chess).

Agent will have a fixed time limit each turn to search for best move. If time limit expires during a player's turn, that player fortifies the match and the opponent wins.

Implementation:

1. Minimax – implemented minimax search.
2. Alpha-Beta – implemented minimax search with alpha-beta pruning.
3. Iterative deepening – implemented iterative deepening on alpha-beta algorithm.
4. Heuristic function (custom_score) – implemented best position evaluation function.
5. Heuristic function (custom_score_2) – implemented alternate position evaluation function.
6. Heuristic function (custom_score_3) – implemented alternate position evaluation function.
7. Heuristic function (custom_score_4) – implemented alternate position evaluation function.
8. Heuristic function (custom_score_5) – implemented alternate position evaluation function.
9. Heuristic function (custom_score_6) – implemented alternate position evaluation function.
10. Heuristic function (custom_score_7) – implemented alternate position evaluation function.

Heuristic functions:

1. Custom_score: Minimizing opponent moves – Heuristic is based on the logic that opponent's move should be minimized as describes in lectures i.e. length (my agent available moves) subtracted by length (opponent agent available moves) multiplied by some factor.
   Factor can be anything ranging from (1, infinity). In this case factor is empirically chosen as 1.5

2. Custom_score_2: Maximizing my player moves – Heuristic is based on the logic that player's moves should be maximized i.e. some factor multiplied by length (my agent available moves) subtracted by length (opponent agent available moves).
   Factor can be anything ranging from (1, infinity). In this case factor is empirically chosen as 1.5

3. Custom_score_3: Maximizing ratio of my agent moves to opponent player moves – Heuristic is based on the logic that player should have more moves in comparison to opponent's i.e. length (my agent available moves) divided by length (opponent player available moves).
4. Custom_score_4: Minimizing ratio of opponent to player moves – Heuristic is based on the logic that opponent should have fewer moves in comparison to player's i.e. –(length(my agent available moves)/length(opponent player available moves))
5. Custom_score_5: Combination of Custom_score_3 and Custom_score_4
   I.e. length (my agent available moves)*length (my agent available moves) subtracted by length (opponent player available moves)*length (opponent player available moves)

6. Custom_score_6: Weighted combination of custom_score_3 and custom_score_4
   I.e. square (length (my agent available moves)) - some factor * square (length (opponent player available moves))
   Some factor ranges from (1, infinity). In this case factor is empirically chosen as 1.5
7. Custom_score_7: Weighted combination of custom_score_3 and custom_score_4
   I.e. some factor * square (length (my agent available moves)) - square (length (opponent player available moves))
   Some factor ranges from (1, infinity). In this case factor is empirically chosen as 1.5

Evaluating Heuristics:

The tournament.py script is used to evaluate the effectiveness of your custom heuristics. The script measures relative performance of my agent (named "Student" in the tournament) in a round-robin tournament against several other pre-defined agents. My agent uses time-limited Iterative Deepening along with your custom heuristics.
The script controls for these effects by also measuring the baseline performance of an agent called "ID_Improved" that uses Iterative Deepening and the improved_score heuristic defined in sample_players.py.
The tournament opponents are listed below:

Random: An agent that randomly chooses a move each turn.

- MM_Open: MinimaxPlayer agent using the open_move_score heuristic with search depth 3
- MM_Center: MinimaxPlayer agent using the center_score heuristic with search depth 3
- MM_Improved: MinimaxPlayer agent using the improved_score heuristic with search depth 3
- AB_Open: AlphaBetaPlayer using iterative deepening alpha-beta search and the open_move_score heuristic
- AB_Center: AlphaBetaPlayer using iterative deepening alpha-beta search and the center_score heuristic
- AB_Improved: AlphaBetaPlayer using iterative deepening alpha-beta search and the improved_score heuristic

Result:

Agent	Performance
AB_Improved	58.6%
AB_Custom	64.3%
AB_Custom_2	70.0%
AB_Custom_3	58.6%
AB_Custom_4	64.3%
AB_Custom_5	64.3%
AB_Custom_6	71.4%
AB_Custom_7	70.0%

All the custom heuristic functions (game_agent.py) perform better than AB_Improved (sample_player.py) by immense margin as depicted in above table.

We would like to opt for AB_Custom_6 i.e. weighted heuristic function because:

1. It outperforms all other heuristic functions with win rate of 71.4% which is higher than all other AB_Improved and AB_Custom heuristic functions.
2. AB_custom_6 heuristic function has higher chances of winning with higher available moves as compared to opponent.
3. It's easier to implement and merely few operations.
4. It's comparable with time taken to calculate other heuristic in consideration.

Output:

```
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  AB_Custom_4  AB_Custom_5  AB_Custom_6  AB_Custom_7
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random       7   3        6   4       10   0        6   4        7   3        8   2        8   2        9   1
    2       MM_Open      9   1        8   2        8   2        5   5        6   4        5   5        9   1        8   2
    3      MM_Center     8   2       10   0        8   2        6   4        9   1        9   1        8   2        9   1
    4     MM_Improved    5   5        5   5        8   2        7   3        6   4        7   3        8   2        6   4
    5       AB_Open      3   7        5   5        4   6        4   6        5   5        4   6        8   2        5   5
    6      AB_Center     5   5        5   5        4   6        8   2        7   3        6   4        4   6        7   3
    7     AB_Improved    4   6        6   4        7   3        5   5        5   5        6   4        5   5        5   5
--------------------------------------------------------------------------------------------------------------------------
          Win Rate:      58.6%        64.3%        70.0%        58.6%        64.3%        64.3%        71.4%        70.0%
```