Building a Game-playing Agent

Alex Staravoitau

alex.staravoitau@gmail.com

Synopsis

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).

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

These rules are implemented in the `isolation.Board` class provided in the repository.

Heuristic Analysis

I have implemented three heuristic approaches that build upon each other gradually improving their performance. Results were verified by evaluating winning performance of agents using each heuristic method on a reasonable amount of game matches. All of them exploit the same idea based on the position of available moves and winning statistics based on those moves. Generally speaking, the closer player position to the board centre the stronger it is in terms of winning, we use this fact as each of the described heuristic methods penalises available moves if those are far from the board centre. Reasoning behind it being that with L-shape moves there are generally less available moves over the next N iterations the closer you are to the board edge.

- **Distance from edges** implemented in `custom_score_v1()`. Calculates the sum of distances of each of the available moves to the board edges.
- **Weighted distance from edges** implemented in `custom_score_v2()`. Adds constant weights to the previous sum, as well as a constant bias for each available move.
- **Dynamically weighted distance from edges** implemented in `custom_score_v3()`. Assigns dynamic weights to the previous heuristic, based on the number of available cells on the board.

Each heuristic calculates player and opponent scores and returns their difference. Described heuristics capture various information about available moves, but generally speaking terminology is as follows: *weight* is the coefficient we apply to the distance from the available move to the edge of the board, and *bias* is a coefficient we apply to the fact that we have a move.

Heuristic #1: Distance from edges

First we try a very straightforward approach of penalising available moves that are further away from the centre of the board, we simply calculate a sum of distances of available moves from the board edges for each player, and then return the difference.

Implementation

Using weights-bias terminology weights are equal to `1` and bias is equal to `0` (Table 1).

```
half_h = game.height / 2
half_w = game.width / 2

for r, c in game.get_legal_moves(player):
    own_score += half_h - abs(half_h - r)
    own_score += half_w - abs(half_w - c)

for r, c in game.get_legal_moves(game.get_opponent(player)):
    opp_score += half_h - abs(half_h - r)
    opp_score += half_w - abs(half_w - c)

score = own_score - opp_score
```

Table 1. Heuristic #1 implementation.

Evaluation

Just as one would expect this obvious approach didn't yield good results and actually turned out to be worse than ID_Improved from the lectures: **67.79%** for ID_Improved versus **65.36%** for Student_v1 (Table 5 in Appendix).

Heuristic #2: Weighted distance from edges

Second approach extends the `custom_score_v1()` function by adding a constant value for each available move to the player and opponent score. After all the fact that we have a move is pretty important and generally carries more information that just a distance from the board edge. Distance, of course, is still a part of the score, although scaled to be in `[0, 1]` range and weighted with a constant coefficients of `0.5` for width and height.

Implementation

Using weights-bias terminology weights are equal to `0.5` and bias is equal to `1`.

```
for r, c in game.get_legal_moves(player):
    own_score += 1
    own_score += 0.5 * (half_h - abs(half_h - r)) / half_h
    own_score += 0.5 * (half_w - abs(half_w - c)) / half_w

for r, c in game.get_legal_moves(game.get_opponent(player)):
    opp_score += 1
    opp_score += 0.5 * (half_h - abs(half_h - r)) / half_h
    opp_score += 0.5 * (half_w - abs(half_w - c)) / half_w

score = own_score - opp_score
```

Table 2. Heuristic #2 implementation.

Evaluation

This heuristic performs much better, and from my initial tests already yields better results than ID_Improved (which never got past 70% on my machine): **68.50%** for ID_Improved versus **70.71%** for Student_v2 (Table 6 in Appendix).

Heuristic #3: Dynamically weighted distance from edges

The main highlight of the third version of our heuristic is dynamic weights that we assign to the fact that a move is present and to its distance from the board edge. As I've mentioned earlier, the fact that we have a move is fairly important, however, it is *more* important if we are running out of available empty cells on the board, and *less* important if we still a lot of non-occupied cells. In order to capture that info we make player score weights and bias dependent on the fraction of empty cells on the board. The more there are empty cells the higher the bias (e.g. the fact that we *have* available move, not *how good* it is).

Implementation

Using weights-bias terminology weights are equal to:

`(blank_cells / total_cells) / 2`

and bias is equal to:

`1 - (blank_cells / total_cells).`

```
c_center = len(game.get_blank_spaces()) / (game.height *
game.width)
c_moves = (1 - c_center)
c_center = c_center / 2

for r, c in game.get_legal_moves(player):
    own_score += c_moves
    own_score += c_center * (half_h - abs(half_h - r)) / half_h
    own_score += c_center * (half_w - abs(half_w - c)) / half_w

for r, c in game.get_legal_moves(game.get_opponent(player)):
    opp_score += c_moves
    opp_score += c_center * (half_h - abs(half_h - r)) / half_h
    opp_score += c_center * (half_w - abs(half_w - c)) / half_w

score = own_score - opp_score
```

Table 3. Heuristic #3 implementation.

Evaluation

This approach performs slightly better than the one with fixed weights and biases: **67.71%** for ID_Improved versus **71.71%** for Student_v3 (Table 7 in Appendix).

Results

Heuristic	ID_Improved	Student
Distance from edges	**67.79%**	65.36%
Weighted distance from edges	68.50%	**70.71%**
Dynamically weighted distance from edges	67.71%	**71.71%**

Table 4. Evaluation results for each heuristic over 200 game matches.

I am using the heuristic based on dynamically weighted distance from edges in my project (`custom_score_v3()`) due to a number of reasons.

- It performs better than heuristic methods from previous iterations and, most importantly, better than ID_Improved, as verified using winning statistics over a reasonable amount of game matches.
- This is the only heuristic that consistently beats ID_Improved against every other player, even after running it over 400 game matches as opposed to usual test of 200 matches (Table 8).
- Out of all candidates described in this analysis the third heuristic captures highest amount of information about the game state that should be useful for winning.
- It is essentially of the same complexity as all other heuristics described here. It does not perform any deep tree traversal and only relies on statistics based on the topmost branching level. Simply put it is not more computationally complex than other heuristics, hence we don't need to justify its usage in terms of time and efficiency tradeoff.

To summarise, scoring methods described here are very basic, and undoubtedly won't beat the world Isolation champion. They are, however, fairly intuitive, can be easily vectorised and seem to represent an iterative process of gradually improving game agent's heuristic.

Appendix

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 160 to 40
  Match 2: ID_Improved vs   MM_Null     Result: 152 to 48
  Match 3: ID_Improved vs   MM_Open     Result: 131 to 69
  Match 4: ID_Improved vs MM_Improved   Result: 117 to 83
  Match 5: ID_Improved vs   AB_Null     Result: 147 to 53
  Match 6: ID_Improved vs   AB_Open     Result: 128 to 72
  Match 7: ID_Improved vs AB_Improved   Result: 114 to 86

Results:
----------
ID_Improved         67.79%

*************************
 Evaluating: Student v1
*************************

Playing Matches:
----------
  Match 1: Student v1  vs   Random      Result: 168 to 32
  Match 2: Student v1  vs   MM_Null     Result: 146 to 54
  Match 3: Student v1  vs   MM_Open     Result: 103 to 97
  Match 4: Student v1  vs MM_Improved   Result: 107 to 93
  Match 5: Student v1  vs   AB_Null     Result: 138 to 62
  Match 6: Student v1  vs   AB_Open     Result: 130 to 70
  Match 7: Student v1  vs AB_Improved   Result: 123 to 77

Results:
----------
Student v1          65.36%
```

Table 5. Heuristic #1 evaluation.

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 154 to 46
  Match 2: ID_Improved vs   MM_Null     Result: 154 to 46
  Match 3: ID_Improved vs   MM_Open     Result: 123 to 77
  Match 4: ID_Improved vs MM_Improved   Result: 124 to 76
  Match 5: ID_Improved vs   AB_Null     Result: 152 to 48
  Match 6: ID_Improved vs   AB_Open     Result: 125 to 75
  Match 7: ID_Improved vs AB_Improved   Result: 127 to 73

Results:
----------
ID_Improved         68.50%

*************************
 Evaluating: Student v2
*************************

Playing Matches:
----------
  Match 1: Student v2  vs   Random      Result: 159 to 41
  Match 2: Student v2  vs   MM_Null     Result: 160 to 40
  Match 3: Student v2  vs   MM_Open     Result: 135 to 65
  Match 4: Student v2  vs MM_Improved   Result: 124 to 76
  Match 5: Student v2  vs   AB_Null     Result: 150 to 50
  Match 6: Student v2  vs   AB_Open     Result: 131 to 69
  Match 7: Student v2  vs AB_Improved   Result: 131 to 69

Results:
----------
Student v2          70.71%
```

Table 6. Heuristic #2 evaluation.

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 163 to 37
  Match 2: ID_Improved vs   MM_Null     Result: 147 to 53
  Match 3: ID_Improved vs   MM_Open     Result: 119 to 81
  Match 4: ID_Improved vs MM_Improved   Result: 129 to 71
  Match 5: ID_Improved vs   AB_Null     Result: 145 to 55
  Match 6: ID_Improved vs   AB_Open     Result: 129 to 71
  Match 7: ID_Improved vs AB_Improved   Result: 116 to 84


Results:
----------
ID_Improved         67.71%

*************************
 Evaluating: Student v3
*************************

Playing Matches:
----------
  Match 1: Student v3  vs   Random      Result: 156 to 44
  Match 2: Student v3  vs   MM_Null     Result: 159 to 41
  Match 3: Student v3  vs   MM_Open     Result: 137 to 63
  Match 4: Student v3  vs MM_Improved   Result: 129 to 71
  Match 5: Student v3  vs   AB_Null     Result: 154 to 46
  Match 6: Student v3  vs   AB_Open     Result: 135 to 65
  Match 7: Student v3  vs AB_Improved   Result: 134 to 66


Results:
----------
Student v3          71.71%
```

Table 7. Heuristic #3 evaluation.

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 321 to 79
  Match 2: ID_Improved vs   MM_Null     Result: 311 to 89
  Match 3: ID_Improved vs   MM_Open     Result: 246 to 154
  Match 4: ID_Improved vs MM_Improved   Result: 247 to 153
  Match 5: ID_Improved vs   AB_Null     Result: 294 to 106
  Match 6: ID_Improved vs   AB_Open     Result: 264 to 136
  Match 7: ID_Improved vs AB_Improved   Result: 240 to 160


Results:
----------
ID_Improved         68.68%

*************************
 Evaluating: Student v3
*************************

Playing Matches:
----------
  Match 1: Student v3  vs   Random      Result: 341 to 59
  Match 2: Student v3  vs   MM_Null     Result: 316 to 84
  Match 3: Student v3  vs   MM_Open     Result: 272 to 128
  Match 4: Student v3  vs MM_Improved   Result: 256 to 144
  Match 5: Student v3  vs   AB_Null     Result: 303 to 97
  Match 6: Student v3  vs   AB_Open     Result: 270 to 130
  Match 7: Student v3  vs AB_Improved   Result: 248 to 152


Results:
----------
Student v3          71.64%
```

Table 8. Heuristic #3 evaluation over 400 game matches.