Artificial Intelligence Nanodegree Project - Build a Game-playing Agent

Deliver

In this project, our mission was to develop an adversarial search agent to play the game “Isolation”. 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). Part 1 of deliver is to modify code in the `game_agent.py`

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.

Steps

To perform the right game, the process was to implement the four functions in `game_agent.py` : - `CustomPlayer.minimax()` : implement *minimax* search - `CustomPlayer.alphabeta()` : implement *minimax* search with alpha-beta pruning - `CustomPlayer.get_move()` : implement fixed-depth and iterative deepening search - `custom_score()` : implement a evaluation heuristic

Analysis

The first evaluation was to guarantee that the code is rightly implemented. Using `python agent_test.py -v` we ran and obtained the following result:

```
test_alphabeta (__main__.Project1Test)
Test CustomPlayer.alphabeta ... ok
test_alphabeta_interface (__main__.Project1Test)
Test CustomPlayer.alphabeta interface with simple input ... ok
test_get_move (__main__.Project1Test)
Test iterative deepening in CustomPlayer.get_move by placing an ... ok
test_get_move_interface (__main__.Project1Test)
Test CustomPlayer.get_move interface with simple input ... ok
test_heuristic (__main__.Project1Test)
Test output interface of heuristic score function interface. ... ok
test_minimax (__main__.Project1Test)
Test CustomPlayer.minimax ... ok
test_minimax_interface (__main__.Project1Test)
Test CustomPlayer.minimax interface with simple input ... ok

----------------------------------------------------------------------
Ran 7 tests in 8.509s

OK
```

Tournament

The `tournament.py` script is used to evaluate the effectiveness for the custom_score heuristics. This script measures relative performance of the agent developed (called “Student”) in a round-robin tournament against several other pre-defined agents. The Student agent uses time-limited Iterative Deepening and the

custom_score heuristic you wrote.

The performance of time-limited iterative deepening search is hardware dependent. These tests were executed in a Intel i7 2.4GHz with 16GB of RAM.

To measure the performance, there is also an agent called “ID_Improved” that uses Iterative Deepening and the improved_score heuristic from `sample_players.py` . For the tournament, our agent has to outperform ID_Improved.

The tournament opponents are listed below. - Random: An agent that randomly chooses a move each turn. - MM_Null: CustomPlayer agent using fixed-depth minimax search and the null_score heuristic - MM_Open: CustomPlayer agent using fixed-depth minimax search and the open_move_score heuristic - MM_Improved: CustomPlayer agent using fixed-depth minimax search and the improved_score heuristic - AB_Null: CustomPlayer agent using fixed-depth alpha-beta search and the null_score heuristic - AB_Open: CustomPlayer agent using fixed-depth alpha-beta search and the open_move_score heuristic - AB_Improved: CustomPlayer agent using fixed-depth alpha-beta search and the improved_score heuristic

Heuristics Developed

To execute the game, 4 heuristics were developed and tested in `tournament.py` and they will be presented below.

Heuristic 1: Difference of Moves

The first one, called `difference_of_moves` is similar to the improved_score heuristic presented during the course and used by `ID_Improved` , except that it uses a more aggressive approach since it punishes the adversarial movement by a multiplier factor. Some factors were tested and the best value, in an empirical analysis, was used here (2.0).

Running the `tournament.py` we obtained the following result:

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

Playing Matches:
----------
C:/Rafael/Estudos&Linguas/MOOC - Udacity/003-Nanodegree Artificial Intelligence/python/AIND-I
solation/tournament.py:100: UserWarning: One or more agents lost a match this round due to ti
meout. The get_move() function must return before time_left() reaches 0 ms. You will need to
 leave some time for the function to return, and may need to increase this margin to avoid ti
meouts during  tournament play.
  warnings.warn(TIMEOUT_WARNING)
  Match 1: ID_Improved vs   Random      Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null     Result: 17 to 3
  Match 3: ID_Improved vs   MM_Open     Result: 14 to 6
  Match 4: ID_Improved vs MM_Improved   Result: 16 to 4
  Match 5: ID_Improved vs   AB_Null     Result: 13 to 7
  Match 6: ID_Improved vs   AB_Open     Result: 11 to 9
  Match 7: ID_Improved vs AB_Improved   Result: 12 to 8


Results:
----------
ID_Improved         72.14%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 18 to 2
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 14 to 6
  Match 4:   Student   vs MM_Improved   Result: 10 to 10
  Match 5:   Student   vs   AB_Null     Result: 15 to 5
  Match 6:   Student   vs   AB_Open     Result: 11 to 9
  Match 7:   Student   vs AB_Improved   Result: 17 to 3


Results:
----------
Student             71.43%
```

Heuristic 2: Final Countdown

The second heuristic gives a different treatment as long as the game is developed. It starts with some aggressiveness factor and, in the final third of the game, becomes more aggressive, enforcing when a move gives a bad score to the adversarial.

This heuristic performed that way:

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 18 to 2
  Match 2: ID_Improved vs   MM_Null     Result: 16 to 4
  Match 3: ID_Improved vs   MM_Open     Result: 10 to 10
  Match 4: ID_Improved vs MM_Improved   Result: 13 to 7
  Match 5: ID_Improved vs   AB_Null     Result: 14 to 6
  Match 6: ID_Improved vs   AB_Open     Result: 10 to 10
  Match 7: ID_Improved vs AB_Improved   Result: 18 to 2


Results:
----------
ID_Improved         70.71%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 19 to 1
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 14 to 6
  Match 4:   Student   vs MM_Improved   Result: 13 to 7
  Match 5:   Student   vs   AB_Null     Result: 14 to 6
  Match 6:   Student   vs   AB_Open     Result: 13 to 7
  Match 7:   Student   vs AB_Improved   Result: 14 to 6


Results:
----------
Student             72.86%
```

Heuristic 3: Run from Adversary

The third heuristic thinks of the game dynamics and tries to keep far from the adversary move. For that, it uses the distance between the pieces as the absolute value in both x and y measures. This is typically a defensive approach.

Then, the goal is to use the sum of current distances as evaluation function.

This heuristic presented the following result:

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

Playing Matches:
----------
C:/Rafael/Estudos&Linguas/MOOC - Udacity/003-Nanodegree Artificial Intelligence/python/AIND-I
solation/tournament.py:100: UserWarning: One or more agents lost a match this round due to ti
meout. The get_move() function must return before time_left() reaches 0 ms. You will need to
 leave some time for the function to return, and may need to increase this margin to avoid ti
meouts during  tournament play.
  warnings.warn(TIMEOUT_WARNING)
  Match 1: ID_Improved vs   Random      Result: 16 to 4
  Match 2: ID_Improved vs   MM_Null     Result: 13 to 7
  Match 3: ID_Improved vs   MM_Open     Result: 13 to 7
  Match 4: ID_Improved vs MM_Improved   Result: 14 to 6
  Match 5: ID_Improved vs   AB_Null     Result: 11 to 9
  Match 6: ID_Improved vs   AB_Open     Result: 11 to 9
  Match 7: ID_Improved vs AB_Improved   Result: 14 to 6


Results:
----------
ID_Improved         65.71%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 18 to 2
  Match 2:   Student   vs   MM_Null     Result: 15 to 5
  Match 3:   Student   vs   MM_Open     Result: 15 to 5
  Match 4:   Student   vs MM_Improved   Result: 12 to 8
  Match 5:   Student   vs   AB_Null     Result: 13 to 7
  Match 6:   Student   vs   AB_Open     Result: 12 to 8
  Match 7:   Student   vs AB_Improved   Result: 12 to 8


Results:
----------
Student             69.29%
```

Heuristic Analysis

Here we have the preliminary results from the heuristics:

Heuristic	ID_Improved	Student
H1: Difference of Moves	**72.14%**	71.43%
H2: Final Countdown	70.71%	**72.86%**
H3: Run from Adversary	65.71%	**69.29%**

Looking for the results, acting defensively in the beginning and being more aggressive in the end (heuristic 2) is the best approach for these ones? It seems one only evaluation doesn't provide enough information to a fair analysis, since some causality could happen. To avoid that, I implemented a variation of tournament program, which runs a match in a 5-set battle (as usually in games like tennis, volley etc), presenting the winner in the end.

Besides, analysing the results primarily, a small modification was introduced in heuristics, to see if privileging the center will give some advantage in the evaluation function, which was mentioned during the classes. This modification follows the code below:

```
directions = [(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)]

off_center = [(r + dr, c + dc) for dr, dc in directions
              if 0 <= r + dr < game.height and 0 <= c + dc < game.width]
player_location = game.get_player_location(player)
if player_location == center:
    bonus = 1.5
elif player_location in off_center:
        bonus = 0.5
```

The following table presents the result from the heuristics:

Heuristic	Match Result	ID_Improved Avg	Student Avg
H1: Difference of Moves	3 - 2 WON	74.42%	73.42%
H1: Difference of Moves + center	3 - 2 WON	67.43%	70.14%
H2: Final Countdown	2 - 3 LOSE	69.57%	68.86%
H2: Final Countdown + center	3 - 2 WIN	68.57%	70.43%
H3: Run from Adversary	1 - 4 LOSE	66.85%	69.71%

As we can see, the results present the heuristic "Difference of Moves" adopting the aggressiveness factor of 2.0. This more aggressive approach, indeed, presented better results against "improved score" executed by the ID_Improved although the factor of 2.0 resulted in a victory of 3 to 2. Keeping more aggressive and bonifying the central position, as implemented in the heuristics named "privilege center" and "final countdown with center", presented the best results in general. In both cases, the results couldn't allow which one was better, since the scores (3-2) and the averages (70.14% and 70.43%) were very close.

Heuristic Chosen

As defined in the project submission, the chosen heuristic was the more aggressive one, privileging the center (in the code, named `"privilege_center"`). The reasons for choosing that are presented below:

1. The numbers were consistently one of the bests in the heuristics. This heuristic defeated ID_Improved in several simulations. Despite the difference between the heuristics can't be statistically proven (in the table above, the same heuristic without), the data presented good results for both `"privilege_center"` and `"final_countdown_with_center"` .
2. Since the results were very similar between both heuristics (as a matter of fact, the difference of both is just a tuning moment to using a more aggressive factor), the second criteria to decide was performance. Since `"privilege_center"` was a little bit simpler than the second heuristic, in a 5-set battle it performed faster, probably because it didn't need to run a if/else command to choose the

aggressiveness factor, nor analysing the number of blank spaces for that. The table below presented the time which each heuristic ran, using the same computer hardware and same environment (e.g. memory available during the execution):

Heuristic	Match Duration (in minutes)
H1: Difference of Moves + center	43.04388556083
H2: Final Countdown + center	59.21595096588

** The final performance presented more than 15 minutes of difference - although is hard to compare the exact environment between these two executions, similar environments provided showed that this difference is significant to keep simpler and choosing `"privilege_center"` - in a TV-event, you can use this difference of time to show nice advertisements. :)

3. Finally, adding points in the evaluation function (adopted in both cases) if the move is next to the center tries to keep our player in the central position, which gives better options to move. As we approach the corners, the L-movements become more limited, and that is generally the reason to try keeping our moves in the center.

Conclusion

The heuristics presented here were put to proof against an adversary and the results were, in majority, consistent in winning. A deeper analysis shows that, despite some randomness of results, having a more aggressive approach in some way delivers better results. Furthermore, the number of executions is still not enough to show conclusive results.

The challenge (and further work) is to define new approaches for improving the results consistently, and also improving the number of matches to have better numbers to analyse.