Heuristic Evaluations

Notes

All tournaments are ran on a machine with i7 skylake CPU with 16 gb ram (2016 macbook pro). All tournaments are ran with unaltered tournaments.py file.

Opponent Move Matching (Mutual)

Compares the similar moves that the current player has against the moves the opponent has on the current game board state. The logic behind this is that if there is a chance that the current player will overtake a potential move of the opponent, that is one less move that the opponent has. This is especially useful if that shared move is the only move availale for the opponent.

Also this scoring function tries to capitalize on the moes that give the current player a surplus move advantage by 10%

With n = num legal moves, this will end up running with O(n) time with the comparisons of legal moves between players.

The final returning logic is as follows:

```
return float((own_moves * .90) * (opp_moves) * (num_matches+1))
```

Spaces Left Compared

This evaluation technique aims to get a score with respect to the amount of legal moves available compared to the amount of spaces available on the board (unbiased). Early in the game on a larger board this technique will hold less weight than during the last moments of the game. The scoring function trys to lead the player towards a move that will result in a higher percentage of available real estate compared to the same of the opponent.

With n = legal moves, this will run in O(n) time.

The main logic is as follows:

```
return float(own_moves/spaces_left -(opp_moves/spaces_left))
```

Further Exploration

This evaluation scoring function will try to reach out one more game state for each legal move. For each extended game state it will add to a running sum the amound of extended legal moves. Hence taking into account not only the legal moves of the current state but also the legal moves of the single level foreseeable future. The logic behind this is that even though all current legal moves may seem equal, there may be a couple that run into dead ends which we may be able to avoid.

The original idea to this was to also compare against the same of the opponent but I couldn't figure out how to forecast a game state for the opponent so I left it as only the current player to maximize the total legal move count for current level +1.

With n = legal moves, this will run in O(n^2) time since it visits next legal moves for each current legal move.

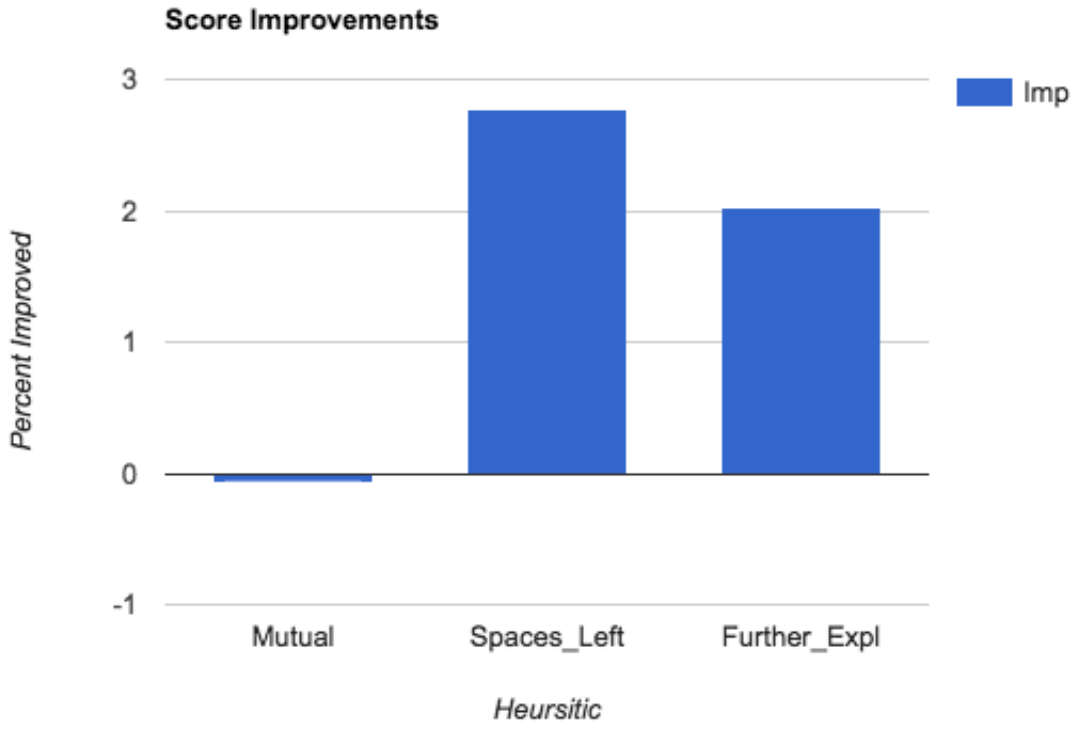
The main logic is as follows:

```
for m in my_moves:
    my_moves_len = my_moves_len + len(game.forecast_move(m).get_legal_moves())
return float(my_moves_len)
```

Conclusion

Below is a table of results for running each heuristic for a total of 4 runs each of tournament.py.

Eval	ID Impr. Avg. Score	Student Score	Difference	
Mutual Moves	74.29%	74.23%	-0.06%	
Spaces Left Compared	74.56%	77.33%	2.77%	<--
Further Exploration	74.65%	76.68%	2.03%	



I would have thought that the further exploration eval function would have beaten the other 2 however it turns out the spaces left heuristic actually was the victor. I have a hunch that since the further exploratino takes into account a higher time to execute, this eats into the ID timeouts and not able to dive into deeper levels. This leads to a valuable lesson that we need to be careful in the time taken in the scoring functions or it just may be counter-productive based on outside factors (such as time limits)

Recommended Evaluation Function

From the results of the test runs, the recommendation that I would suggest is to use a heuristic evaluation function that at least takes the empty spaces of the board in the current state into account. From the tournaments performed, this evalautation function constantly outperformed the opposing players (on average given 20 games). I would also recommend this kind of approach because as the game nears the final moves, the empty board spaces decreases and becomes more and more of a factor in determining victory outcomes. Another reason to incorporate empty states is that the evaluaton function can even go further that just comparing the the number of spaces but also comparing the mutual spaces by both players and empty spaces and capitalizing on moves that lead the current player into an advantage.

Improvements

If further time is spent on this project I would probably look into evaluating any sort of possible symmetry that is

inherit to the game and the players to try to take advantage of reducing time of exploraing further into deeper game states upon scoring.

Also unrelared to scoring I would try to change the way my iterative deepening works in the fact to remember previous depth layer scores and order them in a way in the next depth iteration whereas to make the alpha beta pruning more beneficial.