Heuristics Report

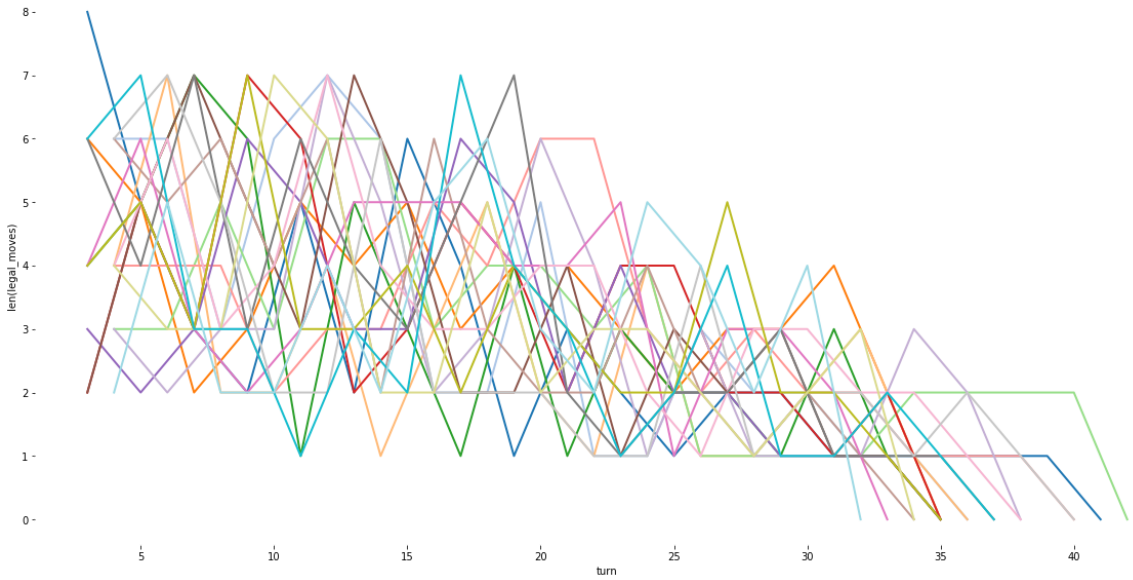
Various methods on heuristics were suggested in the forum and lectures. These included:

- weighting the own options against those of the opponent
- trying to extract symmetry information,
- recognising board states that trivially allow to determine the winner

However, I found that this is hardly working for the knight version of isolation. This is in particular the case because many board states occur where a player has only one possible move, but then has plenty of options in the next one. This is illustrated in the following figure, which shows the number of available moves in each turn for a player for a set of 20 games where both agents followed the same algorithm. While it is true that the number of available moves decayed on average (duh), this is far from monotonous.

In [12]:

```
plot_free_moves(available_moves_lists)
```



Similarly, locking players in is nigh impossible because every 'wall' needs to be two layers thick. As a result, a single layer of additional foresight that returns only information about an imminent win or loss is vastly more meaningful than most of the heuristics that I came up with. I hence tried the following approaches:

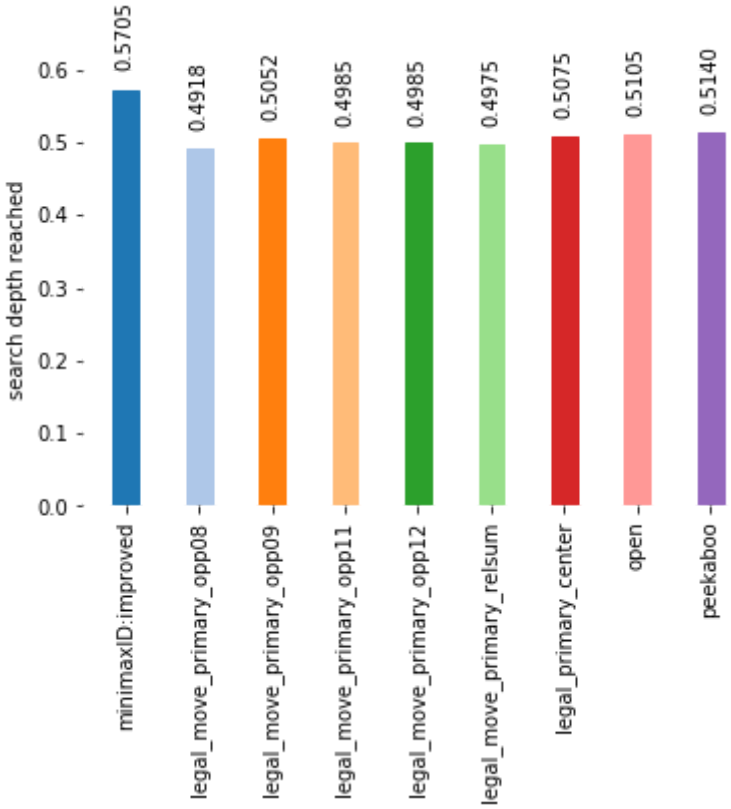
- Weighted difference, i.e., `own_moves - k*opponent-moves`
- Relative difference, i.e., `(own_moves - opponent-moves)/(own_moves - opponent-moves)`
- `improved` heuristic, with an added `center_distance` as a tie-breaker
- Peek-a-boo, a variant of the original AB_Improved, but with a malus if the agents move mean the legal moves of the agent and the opponent now overlap. This is basically an attempt to look into the future, since a legal move is worthless if the opponent can block it. The weight for this malus has been set to less than 1, such that it only applies if the `improved` heuristic for both nodes is identical.

While in any single run of tournament.py there will be overall wins of some of my heuristics over `AB_Improved`, this is not statistically significant. Even in games of AB vs MM, AB occasionally loses (I have tried this both with my code and with other student's implementation of AB and MM), and the following is a

tournament where the custom_score_functions have been set to `improved_score`, `center_score` and `open_move_score`, respectively. The following figure shows the win ration of an `alphabeta:improved` agent against each of these, as well as one minimax player with iterative deepening on the same heuristic.

In [20]:

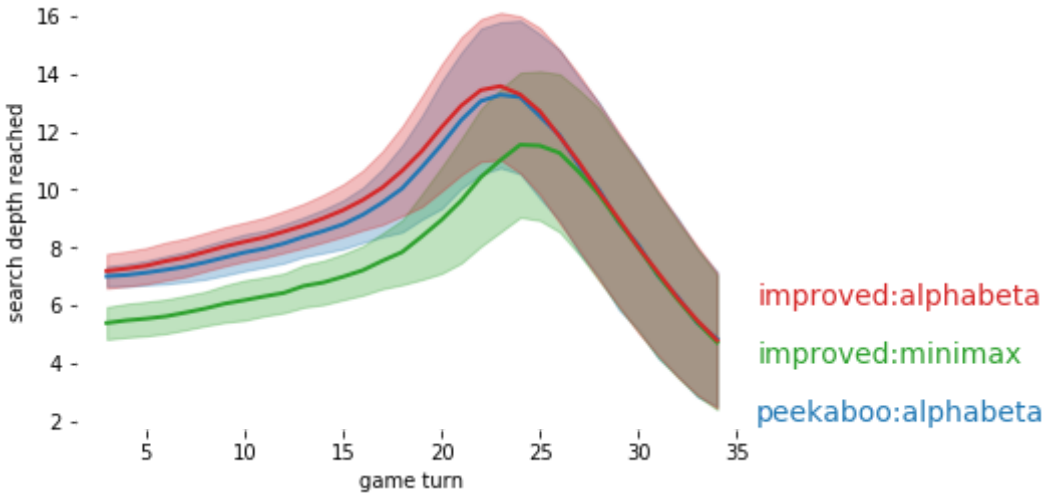
```
plot_win_ratios(opponents)
```



The results show that there is hardly any statistical meaningful difference. However, interestingly the two heuristics that should best the `improved` heuristic (`peekaboo` and `legal_primary_center`) both come out narrowly worse, which is surprising. This could indicate a simple coding error (e.g., giving the opposite advise in tie-breaker cases). However, changing the sign of the tie breaker does not improve the result. The reason is shown in the figure below, which show the mean search depth in each turn for these three agents, plus an envelope given by the standard deviation.

In [19]:

```
last_pos = 0
for i, series in enumerate(search_depth_data):
    if series[0] in ('improved:alphabeta',
 'peekaboo:alphabeta','improved:minimax'):
        last_pos += 2
        plot_range(plt, series[1], series[0], tableau20[2*i], pos=last_pos)
noaxis(plt)
```



Apart from the right flank of the peak (where all agents are able to search the game all the way until its end), the `legal_primary_center` and `peekaboo` heuristics on average reach one less level of search depth during iterative deepening when compared to the `improved` heuristic. This is caused by their more expensive computation. Sadly, this leaves an altered improved heuristic `own_moves - 0.8*opponent-moves` as the best candidate - with a puny win rate of 50.82%

In [ ]:

```

```