# Heuristic Analysis

I will be discussing all the heuristic that I could come up with, however due to hardware limitation ~ 2gb RAM ~ It was difficult to explore complex heuristics. I do appreciate if any better heuristics can be suggested for future implementation:

1. **My Moves**: This strategy just counts the number of current available moves. This strategy can be said as a kind of defense strategy as the user is considered with just it's own move. It's an ok strategy since it doesn't consider the case that it should at the same time be considered with the opponent moves
   ***Code -- len(game.get_legal_moves(player))***

2. **-1*Opponent Moves**: This strategy is somewhat a mirror to the above strategy and focuses on minimizing the Opponent moves. However the better agent will be one that always tries to maximize the difference between the user & the opponent's moves.
   ***Code -- len(game.get_legal_moves(game.get_opponent(player)))***

3. **My_moves - opponent_moves** : This strategy is a mix of the above 2 and is much more optimal than the above 2. It tries to maximize the difference between the current number of User Moves & Opponent Moves. This should be the basic strategy to minimize the opponent's score and maximize personal score.
   ***Code -- my_moves - opp_moves***

4. **My_moves - k*opponent_moves**: This strategy is kind of a variable strategy of Strategy Number 3 -- due to **k** being the variable. This strategy is kind of an offensive strategy since it tries to penalize the players on making moves that grants the opponent opening. The penalization cost increases as K increases. However due to experimentation and hardware limitations I found out that 3 is one of the suitable values for K.
   ***Code -- my_moves - k*opp_moves***

5. **My_moves/total_moves** : This heuristic is one which keeps in mind the number of moves the user has and the total number of available spaces on the board.It tries to maximize a move which increases possibilities or in the worst case reduce the consecutive number of moves in subsequent turns.
   ***Code -- (my_moves / avail_moves)***

```
Results:
----------
ID_Improved          63.57%
```

```
Results:
----------
Student              65.71%
```

6. **(-opp_moves)/total_moves** : This heuristic is a mirror of the above heuristic as it tries to focus more on minimizing opponents moves rather than focussing on it's own.
   ***Code -- (-opp_moves / avail_moves)***

```
Results:
----------
ID_Improved          64.29%
```

```
Results:
----------
Student              70.00%
```

7. **(My_moves/opp_moves) :** This strategy tries to maximize the possible moves for the user and at the same time tries to minimize the number of opponents moves.

```
Results:
----------
ID_Improved          62.86%
```

```
Results:
----------
Student              65.00%
```

8. This strategy involves playing defensively in the start due to large number of available moves and it's tough to decrease the possibilities in the beginning however, later on as the board gets filled it moves to an offensive strategy. This strategy combines the Strategy number 4 & 3 and tries to bring out the most of them.

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

```
Results:
----------
Student              72.14%
```

From the above results it's quite clear the best heuristic is the 8th( 10th one in the game_agent.py) and the next best is Heuristic number 6. Moreover the best agent would be one which tries to maximize the user's score and as well minimize the opponent's score, which the following heuristic captures properly. Moreover the heuristic is simple to calculate and not complex thus allowing the Agent to explore to more depth.