Heuristic Analysis

For the Isolation Game Agent Project I tried five different heuristics which are listed below. I marked with an asterisk (*) the one I decided to choose:

H1: X2 Weighted Score.
H2: X4 Weighted Score.
H3: Center Score.
H4: Proportion Score.
H5*: H3 + H2.

All heuristics were tested using two different computers. I noticed that the agent performed better at times on the computer with the highest processing power:

i7: Linux Ubuntu 64-Bit. 16 GB RAM. Core i7 Processor.
i5: Linux Ubuntu 64-Bit. 16 GB RAM. Core i5 Processor.

Chosen Heuristic

For my implementation I used H5, a combination of H3 for the first moves in the game and H2 for the rest. The results with this heuristic were better than the rest. I thought was a good idea to first choose moves closer to the center, where more positions are available, then when this positions are exhausted and having more chances to move from there we could just use an aggressive weighted score.

Heuristic Tournament Results

H1: X2 Weighted Score

For this heuristic I used the following formula:

own_moves – (2 * opponent_moves)

The overall performance was good, but I stated to notice some cases where my agent did not performed well and even loose a few times against the ID_Improved agent.

Results:

H1	i7		i5	
ATTEMPT	**ID_Improved**	**Student**	**ID_Improved**	**Student**
1	71.43%	79.29%	72.14%	75.00%
2	65.71%	70.00%	70.00%	72.14%
3	68.57%	73.57%	71.43%	77.14%

H2: X4 Weighted Score

For this heuristic I used the following formula:

own_moves – (4 * opponent_moves)

After using the H1 heuristic I decided to start increasing the weight constant. I played with the values until I found a point where the weight was giving me sufficiently good results. Too high values performed badly. Anything below 2 was not good enough. The final decision was to set the weight with a value of 4.

In comparison with H1, H2 offered much better performance. I also thought this could be a good way to approach the Horizon Effect problem in a more aggressive fashion.

Results:

H2	i7		i5	
ATTEMPT	**ID_Improved**	**Student**	**ID_Improved**	**Student**
1	67.14%	75.00%	68.57%	75.00%
2	71.43%	73.57%	70.71%	72.14%
3	72.86%	74.29%	70.71%	72.86

H3: Center Score

For this heuristic I had to first find a way to find the distance between two points in a grid assuming of course that the second point will always be the center of that grid. (Figure 1)

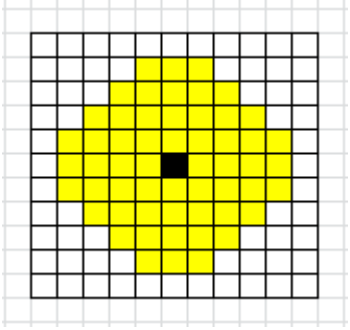

Figure 1: All yellow squares are examples of what could represent a good move. The closer to the center square the better given that more moves are available from the center. However those moves could be only available in early stages of the game.

After doing a quick search I found a formula that could be used on H3 (Figure 2), however the version I used to compute the distance is slightly different.

$$\text{distance} = \sqrt{X^2 + Y^2}$$

Figure 2: Distance of one point to the center of the grid, if the two points are X grids apart horizontally and Y grids apart vertically. The version used on this heuristic subtracts the half of the board width (s=w/2) to X and Y, what give us the following formula: sqrt((x-s)^2 + (y-s)^2).

Therefore, the score function used is:

opponent_distance - own_distance

This heuristic performed better than ID_Improved specially if it was applied at the first stages of the game, but even like that the difference between both agents was very narrow. As I said before, those moves were only available for a limited time. That's why I implemented H3 only at the very beginning of the game.

Results:

H3	**i7**		**i5**	
ATTEMPT	**ID_Improved**	**Student**	**ID_Improved**	**Student**
1	65.71%	74.29%	70.71%	70.00%
2	67.57%	68.57%	67.86%	67.14%
3	64.29%	72.14%	69.29%	70.71

H4: Proportion Score

This heuristic is the proportion of moves available for each player with respect to all current available valid moves. For this heuristic I used the following formula:

my_proportion * 10 - opponent_proportion * 10

Some extra validations needed to be added in order to avoid errors. In order to improve performance of H4 I decided to add a weighted value to the opponents move value. Results weren't much better, so I decided to stick with regular values. I noticed that i5 results were not so good at times.

Results:

H4	i7		i5	
ATTEMPT	**ID_Improved**	**Student**	**ID_Improved**	**Student**
1	65.71%	70.71%	72.86%	77.14%
2	57.86%	73.57%	65.71%	64.29%
3	67.14%	76.43%	67.86%	75.00%

H5*: H3 + H2

This heuristic uses H3 for the first 8 moves (4 for each player) and then applies H2 for the rest of the turns. The formula can be simply described like this:

```
if move_count > 8:
      return H2()
else:
       return H3()
```

Results:

H5*	i7		i5	
ATTEMPT	**ID_Improved**	**Student**	**ID_Improved**	**Student**
1	67.86%	74.29%	67.86%	71.43%
2	70.00%	70.71%	66.43%	75.00%
3	65.71%	73.57%	65.71%	74,29%

Conclusions

The evaluation function was the hardest part of this project. A bad evaluation function can lead to very bad results. The performance of the algorithms is highly dependent on the processing power available. Powerful hardware can significantly improve the results.

New techniques should also be applied. One interesting thing I read about Deep Blue on its paper was the variety of implementations it had. Some of the algorithms were implemented on hardware, some on software and some had both versions. The evaluation function on deep blue was implemented on hardware, I guess to increase performance. Building a super smart machine requires more than just a regular computer and a programming language.