1. Introduction

For this AIND project I have evaluated **three different heuristic evaluation functions** used in Isolation board game. This document discusses each of them separately and provides a simple summary at the end. Each heuristic function was evaluated using the AIND tournament program.

2. Heuristic Evaluation Functions

2.1 Moves function

The first heuristic function is **the simplest of the three**. It calculates a difference between the square of the number of player legal moves and its opponent's legal moves multiplied by 2, a constant chosen rather arbitrarily based on a few experiments. I decided to call this function a "**Moves" function** as it seemed appropriate given how it works. This evaluation function rewards the player for the number of legal moves available to him, but at the same time it slightly penalises him the more the higher the opponent's number of legal moves is. It favours the player's legal moves over the opponent's by squaring the number of his moves by power 2, but at the same time I didn't want the opponent's position to be negligible thus the number of opponents moves are doubled in the returned value. This is a very simple strategy with reasonably low computational requirements which allows for **fast evaluation and for deeper game tree search:** the faster the evaluation the more time there is for the game tree search. Moves function provides a very rough approximation of the current game state rewarding the "freedom of movement" of the player but also nudges it a bit towards chasing the opponent as the opponents legal moves decrease the returned value.

2.2 Edges function

The second heuristic function is slightly **more advanced** than the Moves function. It came out as a result of my experience of playing isolation game in real life against my friends. I noticed that once the knight gets closer to the edge of the board its options to move are limited and drop very low when he gets into the corner of the game board i.e. in between two perpendicular board edges. I decided to model the second evaluation function based on this insight. I decided to call it an "**Edges" function.** The function is designed in a way that rewards the positions away from the game board edges whilst at the same time it penalises the player when the opponent keeps off the edges, too. This in theory should lead to the player pushing the opponent to the board edges, thus limiting the opponents movement options, whilst at the same time trying to keep away from the edges himself. The number of available edge positions in each of the players location serve as weights in a weighted sum where the sum members are all the available legal moves to particular player. To make the player a bit more aggressive against its opponent I doubled the negative weights by multiplying it by 2. The reward is thus directly proportional to the number of available off-the edge moves.

2.3 Defence function

Finally, the last evaluation function focusses on defensive side of the game. I call this function a "**Defence" function**. This evaluation function rewards the distance between the two players as well as the number of open spaces on the board. It does by calculating a distance between the player and the opponent on the board and adding the number of

blank (open) spaces to the result. It aggressively favours the distance from the opponent over the blank spaces: this is achieved by multiplying the players distance by a large, [rather arbitrarily chosen] constant (100). The player using this function should in theory favour the positions that are far from the opponent whilst at the same time trying to search for open areas, too: a classic defensive tactics.

3. Experimental results

All the experiments were performed on Google Container Engine (GKE) [1] running in Docker containers using custom built Docker image publicly available on the Docker Hub [2] which packages my code in it. The source code is also available on Github [3]. The Github repository also contains a Kubernetes job definition [4] which was used to run the tournament jobs on GKE. All of this allows for an easy experiment reproduction. Furthermore, you can find all the results and chart in the Google spreadsheet [5]. Hardware spec used for these experiments was: Intel(R) Xeon(R) CPU @ 2.20GHz, 4GB RAM. I have run 10 tournaments altogether each playing 10 games. This provides for a small statistical sample which can still provide some insight into the behaviour of each heuristic function.

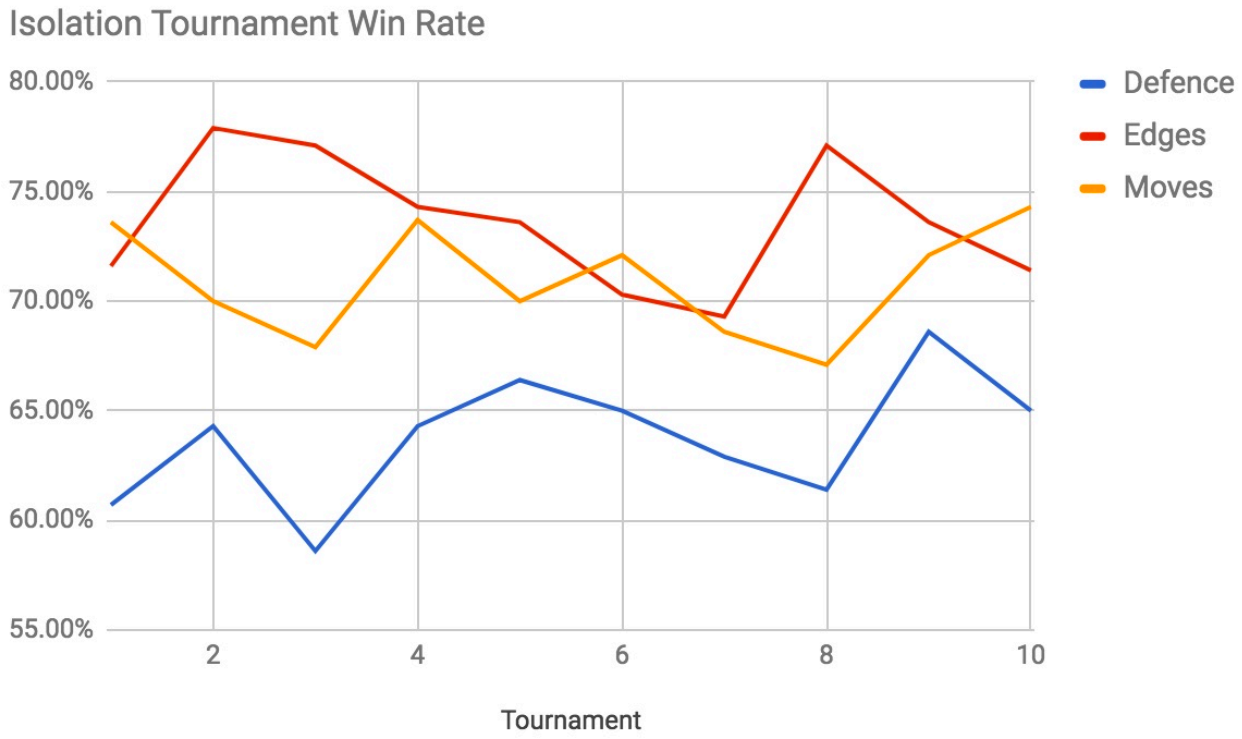
3.1 Evaluation

The Moves function averaged around 70.94% winning rate in 10 tournaments. The overall high win rate is most likely due to the agent being able to perform deeper game tree searches thanks to the function simplicity. Looking at the tournament results breakdown it would appear that this function mostly struggled against the more sophisticated AB opponents which I sort of expected as the moves function is quite simple and thus can struggle when the board gets a bit crowded when more advanced strategies are needed to win. I traded off better strategy for the speed.

The Edge function results seem to show more promising game strategy providing for over 73.62% average win rate and getting over 70% win rate consistently over the course of 10 tournaments. It also does reasonably well against the aforementioned AB agents. Edges function is more advanced and thus we would expect it to do well when the games gets a bit more complex when the game board gets a bit crowded. However, this comes at cost of more computational requirements than the Moves function and thus provides for rather shallow game tree searches. The trade off seems worth it as the player forces its opponents to losses more often than the Moves function does.

Finally, the Defence function performs the worst of the three evaluated functions. It averaged around 63.72% winning rate in 10 tournaments. It's important to note that this function is the most computationally demanding of all three presented functions and thus does not allow for deep game tree search which partially explains the small overall winning rate. Furthermore, the strategy it employs is rather defensive which can lead to the opponent chasing this player until it eventually loses. This function can be the most useful when the player gets surrounded and the only option is to escape regardless of how much computation resources it would cost.

You can find performance of all the evaluated function in the figure below:



4. Conclusion

The results of all the tournaments shown in the chart above suggest that the **Edges function provides the best game strategy**, alas more rigorous tests would have be be done to prove that this is really the case. **The Defence function on the other hadn't gives the worst results,** whilst the Moves function performs reasonably well in fact it is just about worse than the Edges function. This is great because it shows that often the simplest solutions can lead to great results.

The experiment results and their analysis make me think that maybe the isolation game requires using **different strategy at different stage of the game**. Whilst it would be easy to say that based on our results we would recommend the Edges functions because it gives the best results in our experiments, it's reasonably fast to compute [which still allows plenty of time to search the game tree] and is not prone to lack of free space on board [like the other two evaluated functions] whilst at the same time it can also force the opponent to the edges. But I suspect the reality is more nuanced.

The Moves function provides for good evaluation function at the beginning of the game when there is plenty of space for both players it allows deeper game tree searches with "less" strategy which is a trade off we can take at the beginning of the game. We could then transition to the more aggressive Edges function in the middle of the game and favour the strategy over the search and try to push the opponent off the field. Finally if we

get in “trouble” we could try to switch to the Defence function which favours escape at the expense of the shallow game tree search.

References:

[1] https://cloud.google.com/container-engine/
[2] https://hub.docker.com/r/gyre007/isolation/
[3] https://github.com/milosgajdos83/udacity-aind/tree/master/isolation-ai
[4] https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/
[5] https://docs.google.com/spreadsheets/d/1xhwUpERrT8ZT4j-cM_57VzIhZCnGmZ55PfUxPLugbwU/edit#gid=391995746