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.

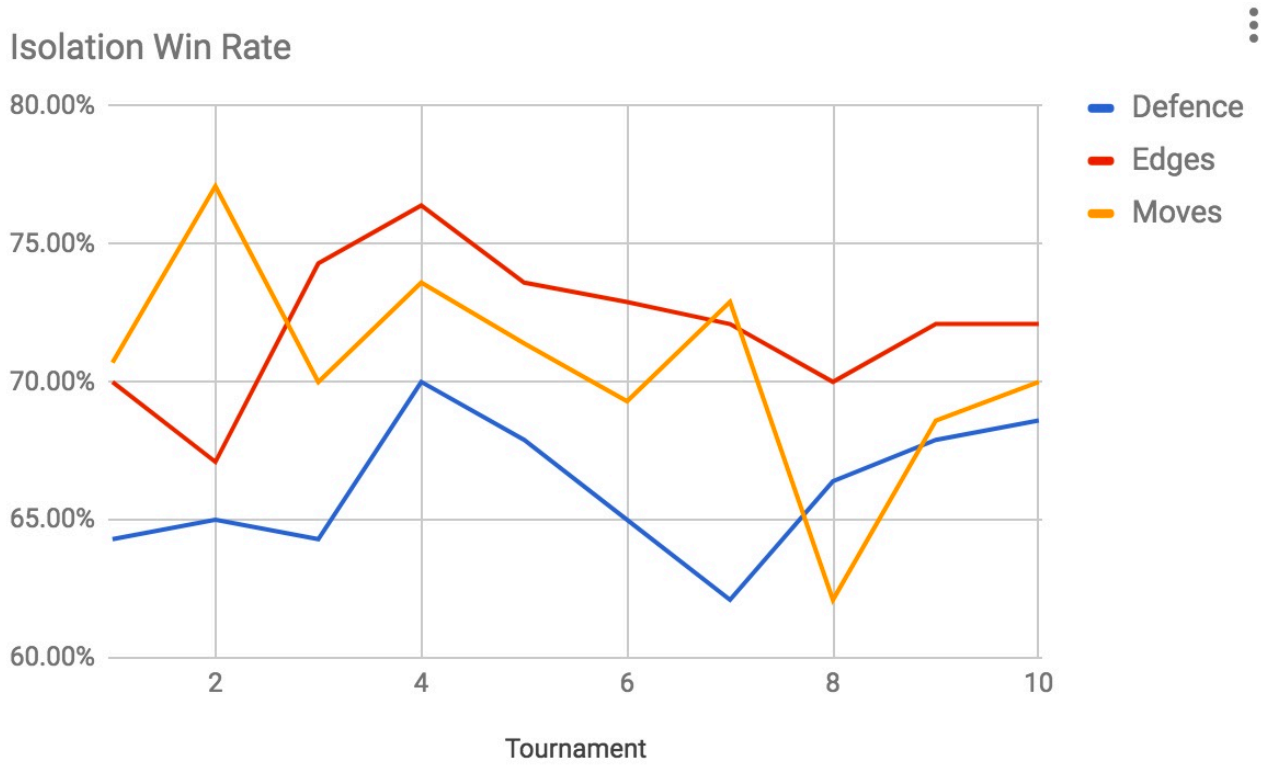
Heuristic Evaluation Functions

The first heuristic function, implemented in **custom_score_3** function, is the simplest of the three. It calculates a difference between the player's and its opponent's number of legal moves. I decided to call this function a "**Moves" function**. This evaluation function is **very simple**. It rewards the player for the number of legal moves available to it, but at the same time it penalises it the more the higher the opponent's number of legal moves is. This is a very simple strategy with low compute requirements which allows for **fast evaluation thus there is more time available for game tree search**. We basically evaluate fast and search the game tree deeper. Moves function provides a rough approximation of the current state of the play rewarding the "freedom of movement" of the player. The moves function **averaged around 70% winning rate** during my evaluation in 10 tournaments. This is a higher success rate than I expected. It would appear that this function mostly struggled against **AB_** opponents. Given the AB_ agents employ similar techniques and evaluation functions the experiments show a very equal battle between these two opponents.

The second heuristic function, implemented in **custom_score_2**, is [in theory] slightly **more advanced**. It came out as a result of my experience of playing isolation in the real life several times against my friends. I noticed that once the knight gets closer to the edge of the board its options to move are limited and often drop to very low when it gets into the corner of the game i.e. in between two perpendicular board edges. I decided to model the second evaluation function based on this insight. I call this function the "**Edges" function.** It 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 could lead to the player pushing the opponents to the edges (i.e. cornering it) whilst at the same time trying to keep away from the edge itself. I've used the number of possible off the edge positions each player has as weights in the evaluation - the reward is thus proportional to the number of possible edge moves. We can look at this function as a weighted sum where the weights are the available legal off the edge moves and the sum items are all the available legal moves. The results seem to show promising game strategy path providing for **over 72% average winning rate** and getting over 70% winning rate consistently over the course of 10 tournaments. This function is however **more computationally expensive** than the Moves function and thus provides for **more shallow game tree searches**.

Finally, the last evaluation function, implemented in **custom_score**, focusses on the distance between the players on the board and the open space. 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 aggressively favours the distance from the opponent over the blank spaces: this is done by multiplying the players distance by a large, rather arbitrarily chosen constant 100. The idea behind this evaluation function was to focus more on the defence, than on the offence: the player using this

function should in theory search for spaces that give it bigger distance from the opponent whilst at the same time trying to search for open areas, too. This function performs worse than the previously discussed function. **It averages around 66.15% winning rate** in 10 tournaments. It's important to note that this function is the most computationally expensive of all three presented functions and thus does not allow for very deep game tree search which partially explains the small overall winning rate.



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 gives the worst results** whilst the Moves function, which rewards number of legal moves and penalises the same metric of the opponent performs reasonably well and it is just about worse than the Edges function. The results seem to suggest that maybe the Isolation game rewards aggressive game play strategies like the one provided by the Edges function more. Whilst the Moves function provides for deeper game tree searches the more sophisticated Edges function seem to provide for slightly better results. Looking closely at the particular tournament games we can also notice that the Edge function on average performs visibly better against the AB_ opponents than the simple Moves function, which probably just met its match in AB_ opponents. The results might suggest that maybe the best evaluation function could combine all three strategies depending on the state of the game. We could start with the Moves function when the game play is open and we can afford deep game tree searches, as the cost of evaluation is low. We could then transition to the more aggressive Edges function and favour the strategy over the search and try to push the opponent on the field. Finally if we get in "trouble" we could switch to the Defence function which favours escape at the expense of the shallow game tree search.

Notes

All the experiments were performed on Google Container Engine running in Docker container which uses a custom built Docker image which packages my code into it. The repository also provides a Kubernetes job definition which allows for an easy experiment reproduction. You can find all the results and chart in the following Google spreadsheet.