CE811

'Hanabi' Bot

Assignment 2 - Hanabi

Registration Number: 1804162

Contents

Summary

Hanabi is a game which consists of a team of two to five players, where their task is to try to successfully place cards to create valid sets of matching colours: each with ascending numeric values. Each player must work as a team to accomplish this, as they are unable to see their own cards. With a limited number of hints available, they must rely on hints given to them from other players regarding their own cards, as well as clues they can gather by knowledge of the other players' cards, the cards in the firework pile, and cards that have been discarded.

I had decided to use the rule-based agent from the seventh lab for the basis of my bot, which I have then worked on to try and improve by introducing new rules, enhancing existing rules, and by introducing a genetic algorithm to produce a high scoring set of rules.

Research that I did about this game before working on my agent was to look at possible rules that could further enhance the existing model. As these rules effectively act as behaviours for the bot, I can experiment with different behaviours to see which are the most effective when used together.

Introduction

The problem was to create a bot that would be able to play the game Hanabi, and that would be able to regularly achieve a high score in the Hanabi Python framework. To do this, an algorithm that was discussed in the lectures and/or labs would have to be used and adapted to fit the purpose and actions available to the bot.

This paper will be explaining what the game Hanabi is and identify important rules that would have to be considered when trying to create behaviours that may increase the bot's chance of success. I will then explain my chosen algorithm used for the bot; describing how the bot decides on the best set of actions and behaviours to get the highest score.

Lastly, I will be explaining the process in which I added this functionality to the bot and describe what I had done to try and improve it at each step and how effective this had been.

Hanabi

Hanabi is a card game composed of two to five players, who must work as a team to complete a common goal. The goal of the game is to successfully complete coloured sets of cards, with each set containing cards that ascend in value (1 -> 2 -> 3-> 4 -> 5). For each colour set (white, red, blue, yellow, green), there are ten cards with different values (three cards with a value of 1, two cards with a value of 2, two cards with a value of 3, two cards with a value of 4, and a single card with a value of 5). This means that the highest score that can possibly be achieved in a single name of Hanabi is 25, with multiple opportunities to build the firework in the case that some lower numbered cards may have been discarded. There are three possible ways in which a game of Hanabi can end:

- If the players manage to successfully create all five sets of fireworks (totalling to the maximum score of 25).
- After the last card has been drawn from the deck, each player is allowed only one more turn. During this turn, no players can daw any new cards as the deck is empty.
- The game is lost if all three red tokens are placed. Each token represents a 'life', and any incorrectly placed card results in a token being placed, and a 'life' being removed.

Unlike most card games, players are unable to see their own hand as cards are viewed 'backwards'; so, whilst each player cannot see their hand, they are able to view the hands of the other players in the game. This means that players are reliant on hints that they receive from other players regarding the cards in their hand, along with some clues they can gather from other players' hands, the current state of the firework sets, or by memorizing the cards within the discard pile.

During a player's turn, they can perform a single action from the following actions:

- They can provide a single hint to another player.
- They can discard a card from their hand.
- They can play a card from their hand.

Players are limited as to the extent in which they can provide hints. Each hint can only consist of one of two things: hinting the colour of specific cards or hinting the number value of specific cards. At the start of the game, there are a total of 8 blue tokens available. Each hint consumes a single blue token, and no hints can be given if there are currently no tokens available. However, discarding a card gains back a single blue token, which can then be used again for hints.

As stated, discarding a card allows for a blue token to be returned to the game; allowing for more hints to be made. When a card is discarded, the player also takes another card from the deck (if possible). However, if all blue tokens are present, then a player cannot discard a card and must perform a different action.

Playing a card can have one of two effects. If the card is valid, then it makes part of the firework set. However, if the card is not valid, then it is placed in the discard pile and a red token is added. After this action, the player also draws a card.

Background

The submitted agent uses a rule-based approach to playing the game, which has been adapted from the code from the seventh lab. This agent contains range of different possible rules and behaviours; from which it can decide its action for the specific turn. This agent functions by sequentially performing these rules from a predefined list; the agent's chromosome. Whilst this chromosome contains numbers corresponding to specific rules described within the agent, it is not required to contain all these rules. As rules are attempted sequentially, some rules may not even be attempted if a guaranteed rule is

assigned early on. Because of this, the chromosome that is generated may not have a higher score in comparison to one with less features, as some rules may not be accessible.

The algorithm used to generate the chromosome that is used by the agent is based on a genetic algorithm. A random chromosome is first generated, before mutations are applied to this chromosome. If the new chromosome performs better than the current best chromosome, then the old chromosome is replaced. This process continues for the specified number of generations, before the best scoring chromosome is returned, which can then be copy and pasted into the agent. The fitness of each chromosome is based on the average score from a specified number of games involving each chromosome; ensuring that any variance in performance is accounted for.

When deciding on what algorithm I would decide to choose for my agent, I came across two algorithms that were often being used successfully: Monte Carlo Tree Search (MCTS) [1] [2], and genetic algorithms [3]. Whilst both showed promise in creating a good agent, I was more comfortable with developing an agent with an estimator based on a genetic algorithm than that of a MCTS algorithm; hence that is what I decided to use.

I decided to use a genetic algorithm based on the Stochastic Hill Climbing algorithm, where each generation contains a single chromosome, and the best chromosome is kept between generations. Each generation, a new chromosome is created by performing a mutation on the stored (best) chromosome. If this new chromosome is better than the stored chromosome, then it replaces it and is used in the next generation. I believe that this is an ideal solution for evaluating my agent, as there may be rules that are required in order for the chromosome to be valid. If offspring in a population-based algorithm do not contain these rules, then they will have an automatic fitness of 0; potentially allowing for less-fit parents to be selected. With this algorithm however, checks can be made to ensure that specific rules are always present after mutations; with a new mutation being applied until a valid one is presented for the generation.

As this agent is more dependent on the behaviours that it can display than the algorithm used to decide on the best behaviours, most of my research for this assignment was based on rules that could improve my agents score, and behaviours that similar rule-based agents exhibited.

Whilst not all rules may result in a higher score being obtained, a wider variety of rules provides a higher chance of it, as behaviours can be less generalised. This allows for actions such to be arranged in a much more varying priority; allowing other players to perform actions that otherwise may not have yet been accessible. However, this does not mean that rules must be very simple. Whilst all rules should be based on different decisions, different criteria may result in a single decision being made.

A useful collection of rule ideas was provided in the course references by Joseph Walton-Rivers [4], which contain many variations of similar rules for different circumstances. These

rules are based on his Java implementation of a Hanabi framework and would help to act as a good starting point to possible rules that could be implemented.

Techniques Implemented

The submitted agent consists of two parts:

- ***rule_agent_chromosome.py***

  This is the actual agent that will be used within the Hanabi game. As this agent is rule-based, this file contains the rules that this bot will use to play the game, as well as how the bot interacts with the framework. Using the chromosome generated from *chromosome_evaluator.py*, it sequentially tries these rules. As only a single action can be performed by the agent every turn, it will perform the first action that is possible of it, before its turn ends. For this reason, there should always be a guaranteed action in the chromosome (such as playing a card), as it is not guaranteed that an agent can discard a card or give a hint on a particular turn.

- ***chromosome_evaluator.py***

  This is the file that generates the best scoring chromosome for the rule-based agent within *rule_agent_chromosome.py*. It uses a stochastic hill climbing genetic algorithm, which performs a random mutation on the best currently scoring chromosome. Over time, this will eventually lead to a high scoring chromosome. However, this could eventually reach a local maximum, where it is not able to vary the chromosome much from the currently scoring one.

The agent's chromosome has its effectiveness judged on its average score for several games of Hanabi. These are averaged due to the cards being randomly shuffled each game, which produces a variety of scores.

The full list of rules this agent implements is as follows:

- Play a card that has a high probability of being playable.
- Give a colour hint to another player.
- Discard a card that has a high probability of being useless.
- Discard the oldest card.
- Play the card with the highest probability of being playable.
- Give a rank hint to another player.
- Discard a card if a duplicate is held.
- Discard a card if that colour's firework is complete.
- Discard a card if it is already present in a firework.
- Play a card if it is a one.

- Play a card if it is a five.
- Reveal any ones another player has.
- Reveal any fives another player has.

Experimental Study

I first went about with developing the genetic algorithm. As we had been given provided some basic rules in the lab code, getting a working algorithm early would allow me to evaluate the performance of each chromosome created after implementing a new rule. For the purpose of this algorithm, the initial chromosome will be a randomised set of rules, with a length of half of the possible rules that are available. This allows the size of the chromosome to remain proportionate to the number of possible rules between tests.

I first had to create the functions which would be used for calculating the fitness of each chromosome, as well as the mutations that I would decide to use. For calculating the fitness of each chromosome, I ran a selected number of Hanabi games with my agent using each chromosome; with the final fitness being the mean score. The code for this was also provided within the lab.

Using the mean was important, as each game of Hanabi randomly shuffles the card for each game. This means that each chromosome's score can vary between runs, and a certain score can never be obtained. However, I also had to be mindful of the time it would take to run the algorithm. As I will be running the algorithm each time a new rule is implemented, a high number of runs in the fitness function will produce a better accuracy, however, will also result in a much longer duration. For this reason, I opted to use 25 runs in my fitness function.

The mutations that I opted to use for the algorithm are as follows:

- **Swap Rules**
  This rule randomly selects two indexes of the chromosome, and swaps the associated values stored at each index. This aims to provide a mutation that can strongly change the structure of the chromosome, whilst limiting the mutation to rules that are already present, allowing for testing of different combinations.
- **Add Rule**
  This rule adds a random rule that is not already present to the chromosome. This allows the algorithm to explore developments on chromosomes by increasing the length of the chromosome and adding new information into the current set. Whilst this makes the chromosome more complex, it provides more opportunity for small changes with mutations.
- **Remove Rule**
  This rule removes a random rule from the chromosome. This allows the algorithm to explore developments on chromosomes by decreasing the length of the chromosome and removing information from the current set. Whilst this makes the

chromosome less complex, it allows the chromosome to make more pronounced changes as each different rule will have a larger proportionate effect.

- **Change Rule**
  This rule changes a rule at a random index in the chromosome, to another rule that is not already currently in the chromosome. This rule aims to try alternative combinations of rules, without changing the overall structure of the chromosome.

These mutations would be randomly chosen with equal probability to ensure that all paths of mutation have an equal chance of being explored. Otherwise, this could result in situations where most generations were performed with only a specific mutation; and others may not have been properly explored.

I then used this algorithm with the rule-based agent provided for the lab. Due to the discussed issue of the cards being randomly shuffled each game, I also decided that I would then test the fitness of the best chromosome separately another ten times, before then calculating the mean score from these eleven tests. The number of generations I will be exploring is 150 as this gives a reasonable amount of time for an effective chromosome to be found. The algorithm will be executed multiple times, with the run with the highest scoring chromosome being used to calculate the average score. Below is the table of results I collected from running the algorithm with the basic rules:

Run	Score
Final Algorithm Output	15.72
Rerun 1	15.2
Rerun 2	15.32
Rerun 3	15.52
Rerun 4	15.36
Rerun 5	14.92
Rerun 6	15.24
Rerun 7	14.88
Rerun 8	15.44
Rerun 9	15.32
Rerun 10	15.2
Average	15.2836

Table 1: Average Score for Chromosome [1, 2, 5, 6, 4]

From these results, we can see that the values presented vary. This is due to how the cards are shuffled for each game of Hanabi. Although each value in the table is an average of 25 games, these games can still drastically vary in order to result in the scores that we see.

Below is also a graph of the scores per generation. As this is a stochastic hill climber, we should see the scores vary almost randomly between low and high values. As the best score increases, this may mean that we will have an increasingly varying score for each generation; as some chromosomes may perform well, but after a small mutation may be useless. However, as the algorithm is still ‘learning’, we should still see a trend of the data increase over time. As these are discrete behaviours, we should expect some large jumps in score.

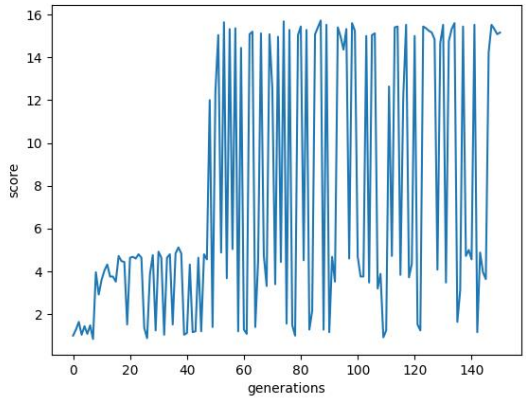

Figure 1: Scores for Each Generation

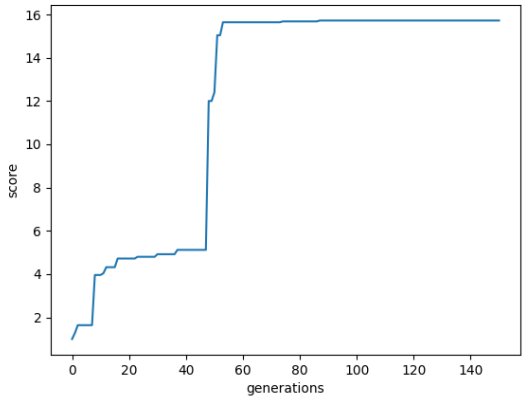

Figure 2: Best Score Found for Each Generation

As we can see from these graphs, the values of the scores for every generation vary wildly. However, we can see a clear trend of the scores increasing over generations; with the increases being mostly sudden jumps in score as much better chromosomes are found.

The first change I wanted to make was to make a couple of improvements suggested by the lab. This was to split rule 2 into two separate rules; to provide a hint for colour and rank separately. I also wanted to edit the function that filters the card list by those that are unplayable; to also look at the discarded cards. If all copies of any rank below the selected card have been discarded, then that card will become unplayable. The table and graphs of the algorithms performance with these additions can be seen below.

Run	Score
Final Algorithm Output	16.12
Rerun 1	14.6
Rerun 2	15.48
Rerun 3	15.72
Rerun 4	15.44
Rerun 5	16.12
Rerun 6	15.52
Rerun 7	14.88
Rerun 8	16.12
Rerun 9	15.6
Rerun 10	15.4
Average	15.5927

Table 2: Average Score for Chromosome [0, 3, 7, 2, 1, 5, 6]

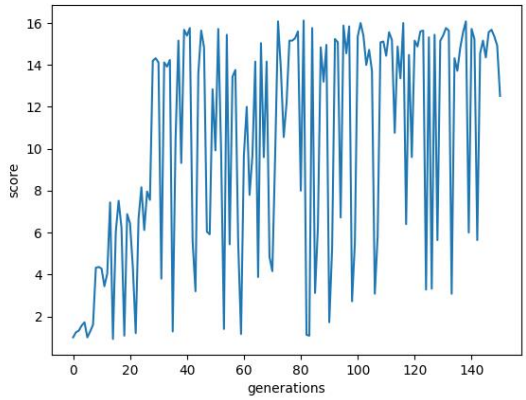

Figure 3: Scores for Each Generation

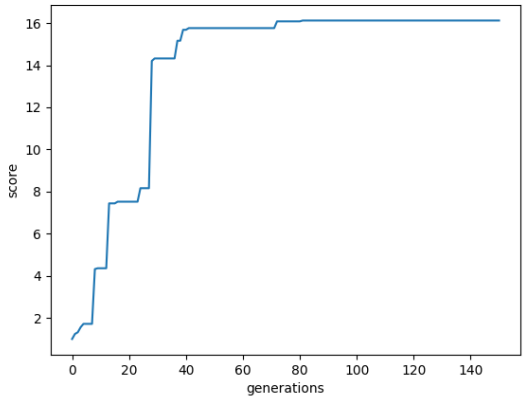

Figure 4: Best Score Found for Each Generation

As can be seen from the results of making the stated changes, we can see that the chromosome has in fact improved. The score that was obtained is higher than the previous score, and the results are more in line with that of the score obtained through the algorithm (16.12 was returned three times).

Next, I wanted to add some rules in relation to how the agent discards cards. We first need to make sure that if the agent currently holds two (or more) of the same cards, then these are discarded as otherwise they will be taking up space in our hand. Other players will also be using out hand as a reference, so we need to have a hand with more options for the other players. Then, we need to make sure that the card in our hand has not already been played successfully. If our card is on the pyramid, then it will no longer be of any use and should be discarded. Finally, we should check to see if the card's colour's firework set has been completed. If so, we do not need any cards of that colour anymore, and the card should be discarded. The table and graphs of the algorithms performance with these additions can be seen below.

Run	Score
Final Algorithm Output	17.24
Rerun 1	15.56
Rerun 2	16.68
Rerun 3	14.88
Rerun 4	15.36
Rerun 5	17.08
Rerun 6	15.6
Rerun 7	15.36
Rerun 8	14.6
Rerun 9	16.16
Rerun 10	14.56
Average	15.7345

Table 3: Average Score for Chromosome [1, 7, 10, 2, 0, 5, 6]

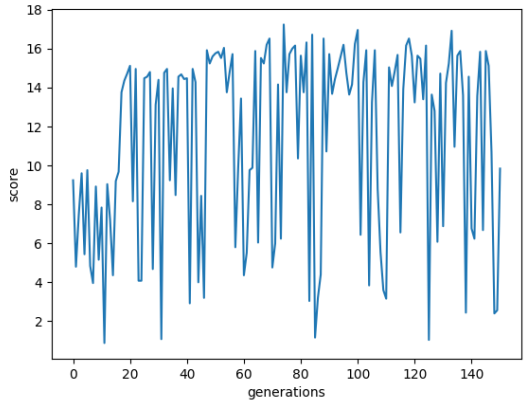

Figure 5: Scores for Each Generation

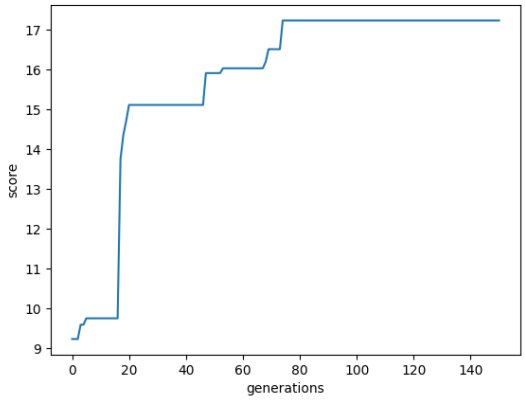

Figure 6: Best Score Found for Each Generation

As can be seen the results of making the changes, giving the agent more opportunities to clear cards from a clogged hand has again improved the average score that was obtained.

I next wanted to add some rules regarding playing cards; specifically playing cards of rank 1 and 5. This is because these cards should always be played as soon as possible, so that they do not waste space in a player's hand. Ones need to always be played as soon as possible, so that other players aren't waiting and discarding good cards for the fact that a specific firework hasn't started. Fives need to always be played, as there is only one copy of each five for every colour; so, if one is accidentally discarded then another cannot be drawn. The table and graphs of the algorithms performance with these additions can be seen below.

Run	Score
Final Algorithm Output	16.96
Rerun 1	16.04
Rerun 2	16.36
Rerun 3	15.44
Rerun 4	15.52
Rerun 5	15.68
Rerun 6	16.96
Rerun 7	15.84
Rerun 8	17.2
Rerun 9	16.76
Rerun 10	17.28
Average	16.3673

Table 4: Average Score for Chromosome [0, 10, 11, 9, 1, 7, 2, 8, 12, 5, 6]

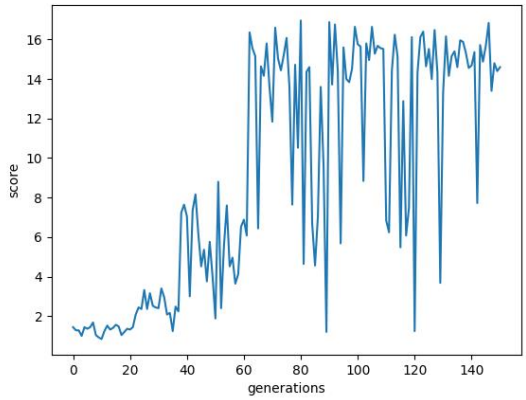

Figure 7: Scores for Each Generation

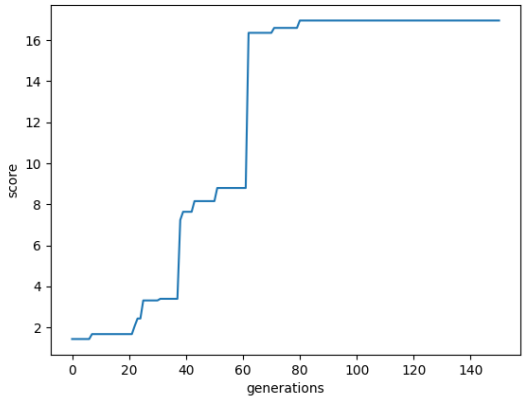

Figure 8: Best Score Found for Each Generation

As can be seen, there is a good improvement to the score after implementing these rules; with both being included in the chromosome.

Lastly, I wanted to add some more rules regarding giving hints; specifically revealing cards of rank 1 and 5 to other players. As with the previous rules, this is because these cards should always be played as soon as possible, so that they do not waste space in a player's hand. However, I need to ensure that players always know what cards are ones, and what cards are fives so that the above rules can be used effectively. The table and graphs of the algorithms performance with these additions can be seen below.

Run	Score
Final Algorithm Output	16.88
Rerun 1	15.56
Rerun 2	16.36
Rerun 3	15.88
Rerun 4	17.4
Rerun 5	16.04
Rerun 6	15.8
Rerun 7	15.6
Rerun 8	16.48
Rerun 9	16.32
Rerun 10	17.12
Average	16.1309

Table 5: Average Score for Chromosome [0, 7, 9, 1, 10, 2, 5, 6]

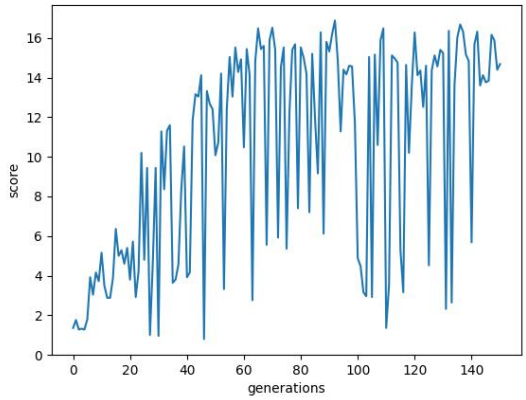

Figure 9: Scores for Each Generation

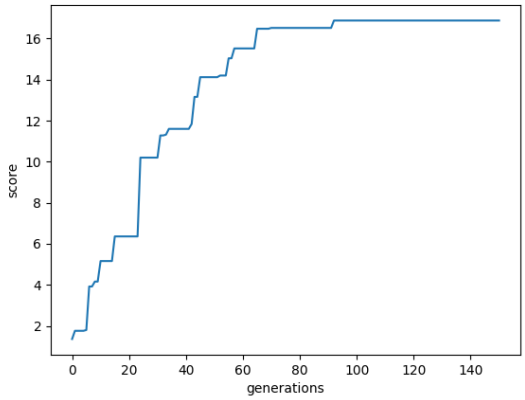

Figure 10: Best Score Found for Each Generation

From the results after implementing these rules, we can see that although the graph shows that the increase was more gradual, the new rules didn't make a positive impact on the score; and weren't included in the final chromosome. This may be due to extra rules introducing multiple more possible local maxima; so, as the algorithm found a chromosome with a good score without the new rules, these rules could end up being left out as mutations occur.

For my submission, I decided to use the chromosome **[0, 10, 11, 9, 1, 7, 2, 8, 12, 5, 6]**, as this produced the highest range of scores, however, did not contain any particularly noticeable outliers. As this also contained the most rules from my chromosomes, this would be able to act differently in a wider range of situations. Below is the result I got from running the '**hanabi-agent-tester**' twice:

	Attempt 1	Attempt 2
Practice	16.0	16.4
Submission	15.9	15.8

Table 6: Table of Submission Scores

Whilst the submission result is lower than my average, it is within the minimum and maximum scores that I had received from running the chromosome within my chromosome evaluator. For this reason, I believe that the agent is performing as expected. Below is the screenshot of the score for **Submission: Attempt 2** from the table:

For Assignment 2, Your Hanabi agent scored an average of 15.8 points per game.

Part #	Outcome	
Class MyAgent is defined	Yes	✔
Agent Score (for Assignment 2)	15.8/25	✔

Figure 11: Chromosome Performance in Submission

Overall Conclusions

One of the main things that surprised me was just how much the score varied between generations. Although I expected it to vary somewhat, I was surprised at how any for some chromosomes, any mutation would result in a vastly lower value. This is likely due to reaching local maximums in my search space however, as if there are no good mutations from the current state of the chromosome, then it will not improve.

With the random nature of the shuffling of the cards, I also found it surprising that scores can vary so much per run for the same chromosome; even though each run is already averaged from a reasonable number of games. I knew that this random shuffling would result in some occasional outliers, so the variance in this must also have been due to how each rule interacts with another. As these tests were performed using copies of the same agent, the same set of rules were playing with each other.

Overall, I believe that my submitted agent performs well as it can consistently get scores of 15 and over, without requiring too many generations to do so. I am also happy with the rules that I have implemented. Although two were not used in the final chromosome, my other rules were able to perform well together. For future work, I would like to look more into retrieving negative hints from the discard pile to further filter if cards are playable or not. This would allow for hints to be drawn, without requiring other players to provide information. In a situation where there are no hint tokens, this may end up in incorrect cards being played or discarded.

I would also like to experiment with implementing a population-based genetic algorithm. Whilst due to the random nature of the game and how the rules act in the chromosome, more restrictions would have to be put on to ensure that specific rules are present in each chromosome. However, this would be an interesting comparison with that of the SHC presented here.

References

[1] J. Goodman, “Re-determinizing Information Set Monte Carlo,” 2019.

[2] P. I. Cowling, E. J. Powle and D. Whitehouse, “Information Set Monte Carlo Tree Search,” 2012.

[3] R. Canaan, H. Shen, R. Torrado, J. Togelius, A. Nealen and S. Menzel, “Evolving Agents for the Hanabi 2018 CIG,” 2018.

[4] J. Walton-Rivers, “Package com.fossgalaxy.games.fireworks.ai.rule,” 09 05 2017. [Online]. Available: http://iggi.fosslab.uk/hanabi/apidocs/com/fossgalaxy/games/fireworks/ai/rule/package-summary.html.