Miracle Merchant AI

Forest Anderson

1 Abstract

This paper introduces an artificial intelligence agent (AI) which tries to achieve the highest score possible in the mobile game "Miracle Merchant". A short-term optimization heuristic is used to achieve a similar score to experienced players. On occasion, it would achieve a score greater than the global high score.

2 Introduction

In Miracle Merchant, the score of a game is determined as the sum of the score of thirteen played hands. Each hand consists of laying four cards down on the table. Points are counted based on synergy between the cards, as well as patterns they make at the end of the hand.

Cards are chosen from four separate piles. Each pile has its own color, and all cards in that pile will be of that color, except for penalty cards. The synergy of each card is randomly generated before it is put in the pile. An example of synergy might be a card that gains extra points if it is placed to the right of a red card due to its preferred color.

One method used was a Monte Carlo approach, where the played cards were randomly chosen and placed. The Monte Carlo approach yielded mediocre results, similar to a player that had just learned the basic mechanics of the game.

The other method consisted of a simple heuristic that optimized the score for each turn. It would go through all of the possible permutations of how to play a single hand to find the one that yielded the highest score. It would then continue the game using that permutation.

There is no way to easily tell the maximum score of a particular game. To find it, you must find the path to that score. This paper examines some strategies to achieve high scores and compares them to the scores of different levels of players.

The methods allow the AI to have global information of the entire game. It knows the order of the deck of cards that will be drawing from, and uses this as part of its input. This has application to the real game because of the "Daily Game" mode. In the daily game, each player in the world receives the same starting decks. This means that the deck order can be recorded, in a first play through. Then, the AI can calculate a high score. Then, the game can be replayed with the path taken to reach the high score.

Information about all of the code used in this code can be found in Appendix A.

3 Method

3.1 Initial Steps and Analysis

The first steps of this project were to replicate the game mechanics in Python. The first step was to look at how each hand was scored. This was complicated as there were many edge cases that were not initially taken into consideration. Unit tests were written at the beginning of the test to prevent regression.

Score had to be calculated as the turn progressed, as the order in which the cards were played would change the score. Optimizing playing order is used as the main strategy of scoring higher.

Next, new games had to be generated for testing. This consisted of analyzing three games from the app. The generation algorithm was found to be as such. First, place the penalty cards in the piles, separated by two color cards. Then, add one of each type of special bonus card to each of the color piles. Finally, fill up the remaining spots with a mix of left-bonus and right-bonus cards.

Next, restrictions had to be placed on what cards could be played. Each of the thirteen hands has a certain color of card that must be played. Near the end of the game, it is possible that you have exhausted the pile of the color that you need, and would then lose the game. Therefore, a list had to be generated that could be used as a lookup table on any particular turn. If a card was chosen to be played, but the turn counter was not higher than the requirement on the lookup table, then the card could not be played, and that permutation would be thrown away.

3.2 Monte Carlo

This first attempt at getting a score was also used to make sure that the system would run properly. The system used a Monte Carlo approach to try and achieve the highest score it could while playing randomly.

It did this by always choosing a random card to play, and then playing it in a random position. This method it would never backtrack on moves played. If it ever receive a score of 0 or less in a hand, it would start a new game.

3.3 Short-term Optimization Heuristic

The second method used a simple heuristic. The system would play all the possible hands for the current turn, and then choose the current hand that had the gave the highest score for the hand. It would do the same for all thirteen turns.

This method included finding all the permutations for the current hand. The number of permutations in a particular hand can be calculated by multiplying the number of piles to choose from by the number of places to play a card. At the beginning of the game, this would calculate as

$$4 * 1 * 4 * 2 * 4 * 3 * 4 * 4 = 6144$$

The AI needed to be able to play a full game while just choosing the permutation that gave it the highest score for that hand. This was done by utilizing the turn restriction lookup table. That way, the AI would always be able to complete the game.

4 Results

4.1 Monte Carlo

The results from Monte Carlo were difficult to obtain. This was because on average only 10 in 100,000 games would be completed until the end. The score obtained from any game could theoretically be as low as 13, and as high as the limit of a particular game. However, these limits are incredibly unlikely, and the average was found to be around 65. The highest score found was 88.

These scores are similar to those of players who have just learned the game, and are attempting to make it to the end without losing. It is a good benchmark to base future scores off of. The calculation time of 100,000 games was over a minute, so it is not a viable option to find higher scores. The memory usage of this method was insignificant. Between games, only the highest score found was saved.

4.2 Short-term Optimization Heuristic

The results from the short-term optimization heuristic were very promising. Games took around three seconds to complete, however they only get stuck in a game around 5% of the time. This made it viable to collect data from.

The data from this method consisted of 4959 sample games. Fig. 1 shows the occurrences of particular scores. The score lowest score recorded was 91, and the highest score was 189. The median of the scores was 133. The 95th percentile was 157.

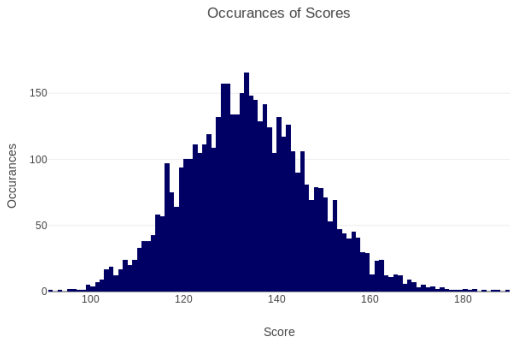

Figure 1: Scores of the Short-term Optimization Heuristic

5 Discussion

5.1 Monte Carlo

The Monte Carlo method was useful for two reasons. First, it was a good way to verify that the system was working properly. It was able to test many edge cases that were not included in the unit tests.

Second, it provided information about what the lower end of the player benchmark would look like. It would rarely make it to the end of the game, and when it did, it did so in a way similar to that of a new player.

With a sample of 100,000 games played, the highest score was 88. This is quite a bit lower than an ideal score. Often times, the AI would only be able to make it halfway into the game and then not be able to go any further. This was due to it not calculating what cards it would need at the end of the game.

There was not an easy method to determine if the Monte Carlo method was going

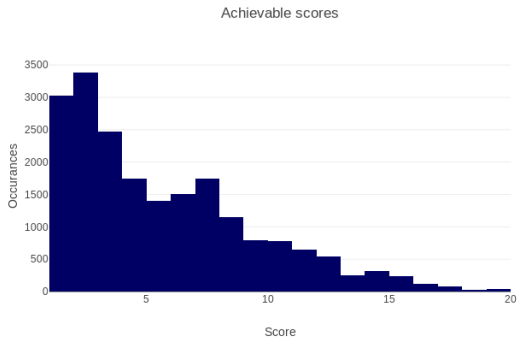

Figure 2: Possible Turn Scores of the Short-term Optimization Heuristic

to get stuck. By picking cards randomly, it was very unlikely that it would progress at all. The condition of requiring a certain color each hand had tremendous effect on this method. Often, the AI would exhaust a pile that it needed later without knowing.

5.2 Short-term Optimization Heuristic

The short-term optimization heuristic method produced great results. The lowest result achieved was greater than the highest result achieved in the Monte Carlo method. The median of this method was greater than the scores achieved by most experienced players. On some of the daily games, it had trouble achieving the high score. This was because the daily high scores were in the 80th percentile of scores reached by the AI, so the AI would only achieve it 20% of the time.

The highest score achieved by the AI was its most impressive feat. In the sample space of 4959 games, 7 of them scored higher than the highest recorded global score of 180. This shows that the AI was a success. It could use some improvement increase the median score achieved, however a different heuristic would have to be used.

As seen in Fig. 2, often the valid scores of a hand were pretty low. The AI would find many hands that were only work one to five points. It would not often find hands that were upwards of twenty points. This shows how much difficulty the Monte Carlo method would have, as it would be more likely to get low scores.

This method only looked at a single turn of the game. This was because the number of permutation increased exponentially as you looked into more turns. The number of permutations of deeper turns can be calculated as

$$6144^{depth}$$

Due to this, there had to be changes made to make sure that the program could continue. Some solutions to this include culling hands that result in lower than 0 score. Another strategy might be to only look at the best n permutations of the current hand, and see how they preform in the next hand. However, with the speed at which this method operates, it it not currently a viable option.

6 Conclusion

The Monte Carlo method was able to achieve scores similar to that of a player just starting to play. The Short-term Optimization Heuristic method was able to achieve an average score similar to an experienced player, and occasionally scores that would rival the highest scores in the world.

In the future, the heuristic could be expanded on further to include information about future turns if the algorithm could be optimized more. The random game generator could also be used to train a neural network to become familiar with the patterns in the game. This could lead to more in depth analysis of particular strategies and moves that an AI could make.

Appendix A Code Repository

All of the code referenced in the paper can be found on this public Github repository:

AngelOnFira/Miracle-Merchant-AI

To examine the notebook with Github's Jupyter Notebook viewer, you can follow this link:

Project Notebook

Note, the notebook may take a few seconds to load.