COSC343: Assignment 1 report

Ben Knox (3938210)
August 8, 2023

1 Introduction

The aim of this assignment was to create an agent to solve the mastermind game in 5 guesses or less, while performing in a reasonable time. The mastermind game is a game where generally there are 4 slots and you must guess each colour in each slot correctly. Once a guess is made, for example 'BBRR' feedback is returned in the form of coloured pegs. A black peg means that there is a correct colour in the correct place, and a white peg means there is a correct colour in the incorrect place. For example if the hidden code was 'BGGB' and our guess is 'BBRR', The feedback received would be 1 black peg, and 1 white peg.

I found that this problem had been solved by Donald Knuth, who solved the mastermind game in guaranteed 5 moves, for the game set up with a code length of 4, and with 5 possible colours to guess from. In our case the settings of the game were not so rigid, and test cases had to be considered for games with a code length up to 5, and possible colours up to 6.

2 Approach

For my solution, I use the same method Donald Knuth had used, but with different optimisations needed for faster running time. This approach involves reducing the number of possible codes, as well as scoring each potential guess and choosing a guess which is likely to give us the most information. This guess may not always be a possible solution to the game based on the information received so far, but will guarantee a solution in the minimum moves possible.

3 Methods Used in This Approach

3.1 Initial guess

My approach was initially to look into the work of Donald Knuth, I found a general explanation of his method here [1], in the section 'Worst case: Five-guess algorithm'. From here I began with the initial guess, as per Knuths' paper [2], he chooses an initial guess for a game with code length 4 of '1122'. In his paper Knuth explains this guess is necessary in achieving 5 moves or less with this method. From here I started with a hard coded initial guess of 'BBRR' the colour equivalent of '1122'.

3.2 Evaluating Each Guess

Upon making a guess in the mastermind game, the opponent player will evaluate this guess and return feedback in the form of black and white pegs. In order to score guesses without actually making them, an evaluate guess method was needed. This

method would compare a guess and a target and return the black and white pegs based on these two codes.

3.3 Pruning/Filtering

The first obvious step of slimming down the number of possible codes is to remove any code that has been guessed from the total set of all possible codes. Then, with the help of a question answered on stack overflow [3], The set of possible codes can further be slimmed down significantly based on the feedback received from each guess.

Firstly, we make an initial guess of 'BBRR', which gives feedback of say 2 black pegs, and 1 white peg. From here, we can remove from the set of all possible codes, any code which when compared to our previous guess 'BBRR' does not give feedback of 2 black and 1 white.

3.4 Minimax

The general idea for this method is that it returns a list of codes which are possible next guesses.

The minimax function works as follows: For each possible solution, the agent calculates the maximum number of remaining possibilities that could result from using a certain guess. Any guess that then has the minimum of this maximum number of possibilities is then added to the next guesses list. So even in the worst case, the guess will provide a large amount information.

Because the minimax function gives a score to each code in the game[1], and not to each code possible at this point[2], some of the codes in next guesses may not be possible answers to the game, hence the implementation of a choose next guess method.

3.5 Choose Next Guess

Some codes in the next guesses list which is returned from the minimax function are not actually a possible solution to the game at that time. In the choose next guess method we prefer to choose a next guess which is a possible solution because this gives a chance of completing the game in that move. However sometimes the best guess will not be a guess which is a possible solution. We still prefer this guess to any code in the list of possible codes because even if a guess may not give the correct solution in this turn, it will give the agent the most information, allowing it to guess correctly in the next turn.

4 Initial Results

After Implementing these things, the agent could play the mastermind game while scoring an average of 4.132 guesses. This can be seen below in table 1, which shows every row as an average of 100 games.

[1]Each code in the game = Every possible code that can be made with the given colours and code length.

[2]Each code possible at this point = The list of codes which has had any impossible codes filtered out.

code length	No. of colours	seed	no. of games	time (s)	average no. guesses
4	5	None	100	25	4.1
4	5	None	100	24	4.1
4	5	None	100	24	4.16
4	5	None	100	24	4.12
4	5	None	100	22	4.09
4	5	None	100	22	4.12
4	5	None	100	23	4.13
4	5	None	100	24	4.19
4	5	None	100	21	4.18
4	5	None	100	28	4.13
Averages:				23.7s	4.132

Table 1: Averages per 100 games

For the game settings of code length 4, and number of colours 5, this worked well. However when moving on to code length of 5 and number of colours 6, it was clear that this either needs significant optimisation or a new solution would need to be found. When trying to run at these new settings, time to complete 100 games was anywhere from 30 - 45 minutes, significantly exceeding the goal of 10 minutes.

5 Optimization

5.1 A General Initial Guess

It was clear that with the number of different code length and colour combinations, some way to generate an effective initial guess was needed. What I chose somewhat leaned on the work of Knuth for his initial guess in the algorithm for code length 4. I determined my initial guess to be 2 of each colour until the code length is reached, with certain checks which will alter this pattern if say there is not enough colours to fill this pattern with respect to the chosen code length.

5.2 Memoization

The problem with my current implementation, for larger code lengths and number of colours, was that large amounts of time was spent in the minimax method, especially for the first guess.

My initial thought was to save the result of every call to the evaluate guess method, so save every code combination with the feedback that it would return. This proved to use more memory than I was happy with, and was not going to be the best option for optimizing this code.

My second idea looked into the fact that there was an obvious bottle neck, the minimax method called after the first guess. The inefficiencies only really appeared after the first guess because no pruning to the list of possible codes had been done at this point. All subsequent guesses are almost instant regardless of saving their results.

I realised that since the initial guess was always the same, any feedback received after this initial guess would provide you with an identical list of possible codes. Given the

list of possible codes is the same, the result of the minimax method would also be the identical, and would directly relate to the feedback of the first guess. So, after the first guess for every round, the feedback is saved as a key, while the list of next guesses returned from the minimax method is saved as the value to this key.

So, every time the agent sees a new combination of feedback after the initial guess, the list of next guesses is saved in a dictionary for use in following rounds.

6 Results

After these optimisations my agent was able to successfully achieve an average score of 4.868 over 1000 games and average time taken of 7 minutes and 26 seconds, for the game settings of code length 5, and colours 6. As shown below in table 2

code length	No. of colours	seed	no. of games	time	average no. guesses
5	6	None	100	6min, 4s	4.75
5	6	None	100	8min, 41s	4.85
5	6	None	100	6min, 44s	4.92
5	6	None	100	8min, 3s	4.98
5	6	None	100	8min, 6s	4.88
5	6	None	100	7min, 8s	4.76
5	6	None	100	6min, 47s	4.85
5	6	None	100	6min, 57s	4.87
5	6	None	100	7min, 35s	4.86
5	6	None	100	8min, 23s	4.96
Averages:				7min, 26s	4.868

Table 2: Averages per 100 games

With these optimizations, the average time for code length 5, and number of colours at 6 dropped from around 40 mins to an average of 7 mins and 26 seconds. As well as this, on testing the case with code length at 4 and number of colours at 5, the average time to run 100 games had dropped from around 23 seconds to around 4 seconds.

Overall this agent performed well in terms of average number of guesses, however, when code length or number of colours is increased the time taken for this agent to run is nearing the acceptable limit. For any game settings beyond the scope of this assignment this agent may perform inefficiently. Despite this, the agent tested on random seeds, does perform within the expected time and guess constraints of the assignment.

Important to note that the more games run in succession, the faster the average time for each game due to the memoization implemented. A result for each possible key is generally stored within the first few minutes of the agent running. After all of these results are stored for use, each game can run at an average of 2 or 3 seconds.

7 Conclusion

I have learned a lot during this assignment, including the use of python dictionaries to store cached results. As well this I struggled through learning how the minimax

method operates. Although I have not implemented minimax recursively, I believe the knowledge I gained when learning this function means I am now well equipped to use minimax with ease in another project.

I also believe that my choice in optimisation for the minimax method is a well balanced optimisation which accounts for both space and time complexity while still achieving the goal of this assignment.

References

[1] Wikipedia. *Mastermind* Mastermind

[2] Donald E. Knuth. *The Computer as Master Mind*. J Recreational Mathematics, Vol 9(1), 1976-77 The Computer as Master Mind

[3] Unknown. *Donald Knuth Algorithm Mastermind*. Stack Overflow Donald Knuth Algorithm Mastermind