Mateo Fiol, Alejandro
20/06/2021
Testing task - GoodAI

Mastermind - A deeper view to the algorithms

In the unity project there are two different approaches for the mastermind AI. First, an AI concept using basic operations over our guess and evaluate it through the fitness function, and keep with the best fit. Second, a genetic algorithm which uses a population in order to reach the solution after N generations.

First approach

I coded the first algorithm thinking how humans would get the proper answer as soon as possible.
When we make a guess and we receive the bulls and cows value, we start permuting the numbers on our guess and making some changes. When in a concrete position our bulls and cows value is lower, it means that the value placed isn't correct and we would step back with our previous guess and keep doing some changes over it, however, if the new code is better than the previous, the process continues with it. Once we know our cows value is zero, we would start only changing the number on one position instead of permuting the values. We would keep this process until we reach the answer.

Implementation

I made a 2d array of four columns and ten rows. Rows refer to the position in our guess and columns the possible values per position (zero to nine). If there is a zero in that position of the array, it means that the value could fit there, however, if there is a -1 that means it can't be there because it lowers the bulls & cows values.

For example:

OUR GUESS → 1785

[0,0,0,0,0,0,0,-1,0,0]	1
[0,-1,0,0,0,0,0,0,0,0]	7
[0,0,0,0,0,-1,0,0,0,0]	8
[0,0,0,0,0,0,0,0,-1,0]	5

We can visually see that 1,7 and 8,5 permuted and didn't increase the b&c value, so we place a -1 in the array in its proper position.

The algorithm starts doing a random guess and evaluates if it fits the user's code. If it doesn't, the algorithm does two operations in this order: permute if the cows value is higher than zero, otherwise mutate some value from a random position of our guess.

Permute	**Mutation**
1785 → 1587	1785 → 1735
Swaps position 2 and 4 values	Changes position 3 value

Selected values in these operations can't be on the array with a -1 in its position, so if it is the case, we keep getting new values randomly until it doesn't. On these operations, if the resulting new code has a higher bulls and cows result than the previous guess, it replaces the selected code (in which we will make further changes). However, if it isn't the case, we place a -1 in the selected position from our array to indicate that value doesn't fit our desired code.

Possible improvements

- In this approach we did only one operation per guess, we could maybe combine the operations in concrete situations.

Genetic evolution algorithm

The idea is to simulate the natural selection of the evolution of Darwin: we have a population made of random guesses and we generate childs using operations over them. The operations are:

- Crossover: select randomly a parent's value per position for the resulting child
- Mutation: changes a random position its value
- Permute: swaps values in random positions

Crossover

Selected parents	[7]4[0]2	3[1]9[8]
Resulting child	7108	

After generating the childs, we evaluate each of them to our user's code and we only take half of the best childs for the next generation. After that, we take the best code and see if it is the correct answer, and if it isn't we keep doing this process over and over.

Thoughts and possible improvements

- This approach isn't the best solution since it takes a lot of generations (memory and time) to reach the solution but I thought it was worth the try.
- In this approach I did every operation full random: it doesn't take into account the previous values per position as I did in the other algorithm.
- Each child generated is a comparison with the user's guess, so it does a lot of recalls to b&c results.
- Further generations could vary depending on the elitism approach taken. Mine was taking only half of the population but could be better to take even less.