Simulator

For sequence of 100 actions and map from *Easy on the right* test my agent can perform on average 7012 simulations.

RHEA

Unfortunately, I didn't manage to implement the RHEA. I've tried to implement a basic evolutionary algorithm just to see how many test cases I can solve using this simpler approach and as a practice before writing RHEA, which eventually didn't happen.

My algorithm passes only one test case from CodinGame - *Easy on the right*. For *Initial Speed Correct Side*, *Initial Speed Wrong Side* and *Deep Canyon* my lander is able to reach the landing side but with too high vertical velocity.

Every game iteration my algorithm does as many generations as it can in the CodinGame's time limit. Finding a new generation for population size N and mutation parameter M consists of four steps:

1. Select N/2 individuals (parents) using roulette wheel selection.
2. Randomly mutate M genes of every parent in order to create a set of children. Gene consists of two numbers: rotation and power. Mutating a gene selects new random values for rotation and power.
3. Evaluate children.
4. Merge original population (N individuals) and children (N/2 individuals). Sort them by their fitness values and select N best individuals which will make up the new population.

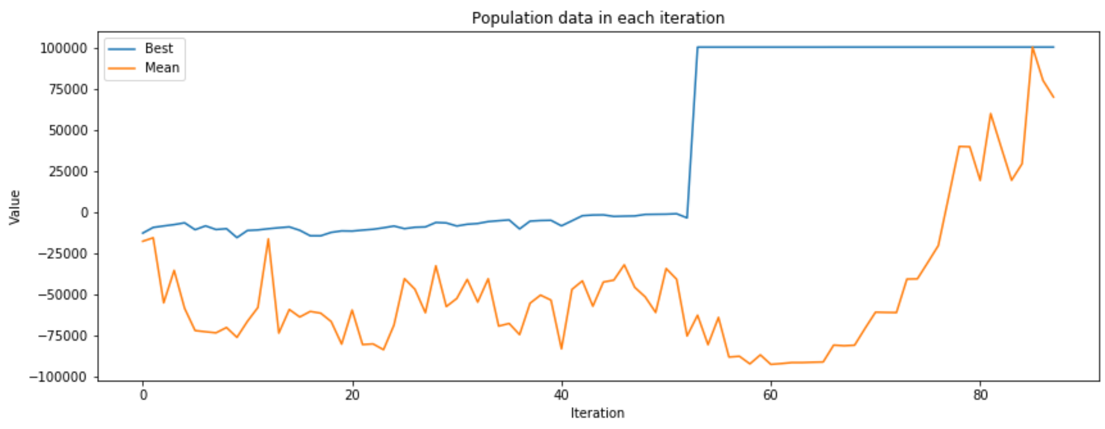
The move I choose is the first move of the best individual from newly created population. During the next iteration I will reuse the population I've calculated. To do so, I delete the first gene (move) of every individual from the population and I'm randomly choosing the last move for every one of them.

Fitness function I use works in the following way:

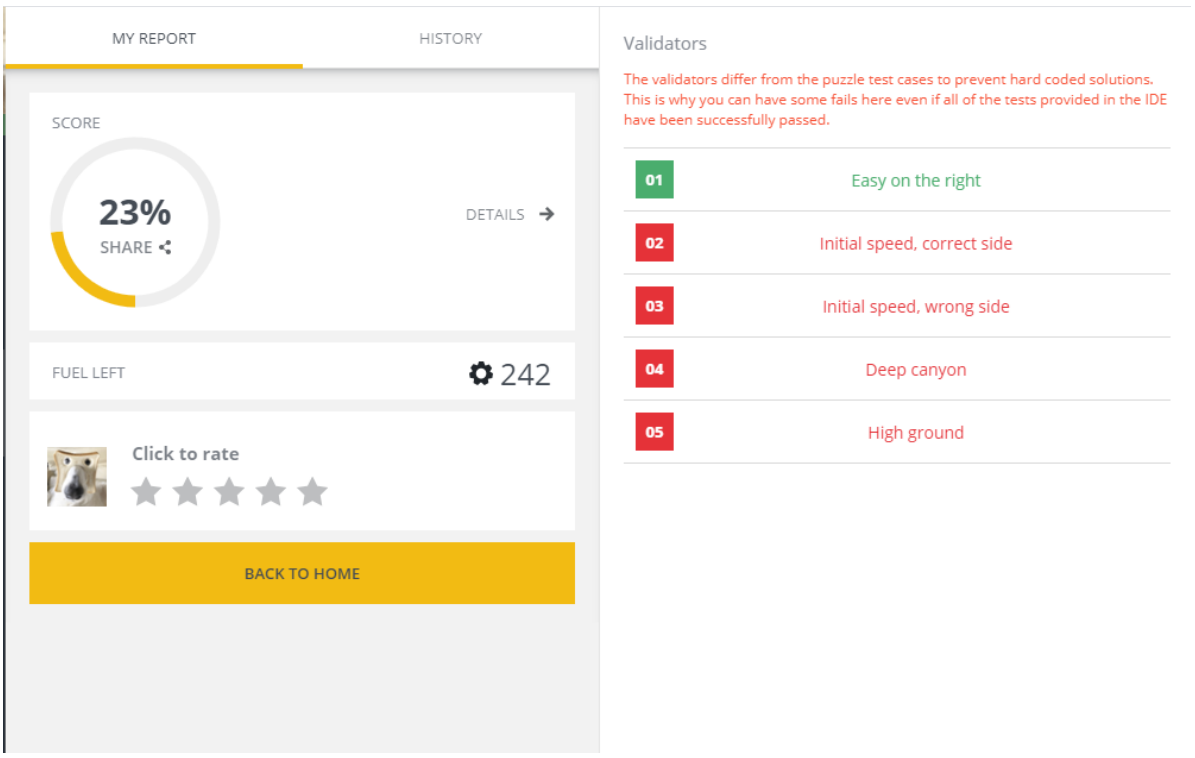
```
LANDING_BONUS = CRASH_PENALTY = 100000
Manh_dist = Manhattan distance from the lander to the center of the landing zone
result = 0
if landed successfully:
    return LANDING_BONUS + remaining fuel

result -= abs(rotation) * 20
if too high vertical speed:
    result -= abs(vertical_speed) ** 2.1
if too high horizontal speed:
    result -= abs(horizontal_speed) ** 2

if crashed on the landing zone:
    return result - CRASH_PENALTY
if flew outside the map:
    return result - CRASH_PENALTY - (Manh_dist * 10)
if still in the air:
    return result - (Manh_dist * 10)
```

Parameters I use are: population size 20, genotype's length 80, mutation parameter M: 10.

Best and average fitness for *Easy on the right* test:



Score on CodinGame:



The validators differ from the puzzle test cases to prevent hard coded solutions. This is why you can have some fails here even if all of the tests provided in the IDE have been successfully passed.

01	Easy on the right
02	Initial speed, correct side
03	Initial speed, wrong side
04	Deep canyon
05	High ground