Advanced & Learning Game AI
ES & NN AI Report

Group members:

Huang Zou
Hantian Jiang
Hongyang Lin
2/28/18

Table of Contents

Approach

Neural Network:

Structure

- 3-Layer Neural Network with 2 hidden layers
- Each of 2 hidden layers contain 12 neurons
- Sigmoid and activision function will apply to inputs to each neuron.
- Weights are evolved in ES

Inputs

- Level scene (19x19)
- Enemies (19x19)
- Mario state (12)
- Buttons pressed in last frame (6)

Outputs

- Buttons (6)
    - Enforce left key and right key can't be active at the same time

Input Strategy

- Strategy 1
    - 8x8 level scene
    - 8x8 enemies
    - 11 mario state
    - Buttons pressed in last frame
- Strategy 2
    - 5x5 level scene
    - 5x5 enemies
    - 6 mario state
- Strategy 3
    - 6 mario state
    - Buttons in last frame
    - Wall ahead
        - Is there wall ahead
    - Enemy ahead
        - Is there enemy ahead
    - Stuck

        - Is mario stucked
    - Jump too long
        - Is this long jump
    - Gap ahead
        - Is there gap ahead
    - Enemy height
        - How height is the nearest enemy

Evolutionary Strategy

Implementation

The ES uses (p/r + c) evolving pattern. The detailed steps are described below:

1. Initialize the population with random weights.
2. Sort the population by fitness.
3. Choose the first ⅓ best population as parents.
4. Mutate the parents to get the second ⅓ population.
5. Recombine the parents to get the third ⅓ population.
6. Sort the population by fitness.
7. Repeat step 3 ~ 6.

Note: We also tried another ES that in every generation ¼ random generated agents join the population. The result will be described later.

Crossover

We used two methods to select parents.

1. Recombine pairs of parents in order
2. Randomly selects parents to recombine

We used three recombination strategies.

1. Randomly crossover weights
2. One point crossover
3. Two point crossover

Mutation

We Mutate weights with Gaussian random float between -1 and 1. In one experiment, we tested with three the initial mutation values (0.1, 0.3 ,0.5).

Fitness function

Progress in the level is given the largest weight in the fitness. Killing enemies will also be significantly reward the agent. The agent will be penalized if it does not jump in front of walls or holds jump button for too long time. The agent is also encouraged to stay in the air.

Formula:
Fitness =
distance passed * 100
+ enemies killed * 500
+ air time
- wall_not_jump time
- jump_too_long time

Experiments and Result

First-Time Data Collection

Two Point Crossover (The Old Fitness Function)

Fitness Function:

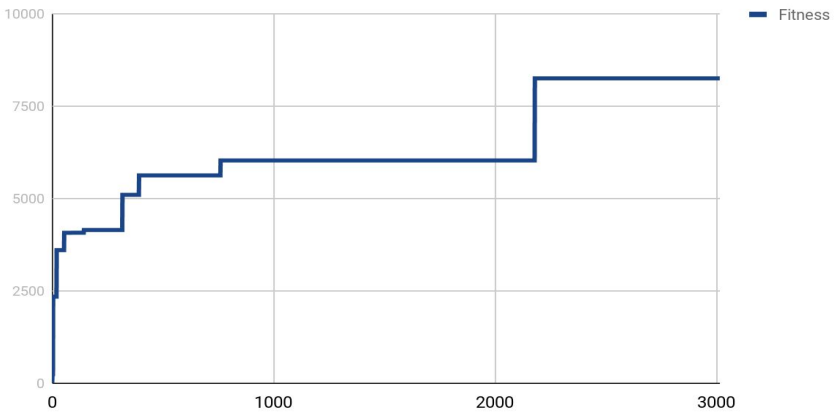
Strategy: Ignore coin value, enhance distance reward, penalize long time immobilization.

Formula: Weighted fitness + Distance * 3 - Collision * 10 - TimeSpent * 5 - Coins * 16;

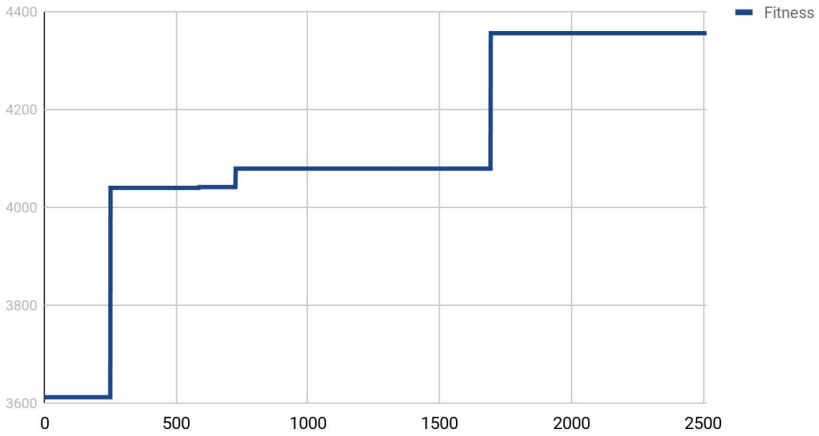
Mutation parameter: 0.3, 0.8, 3;

Result: Level 0 reached about 3/4 of the total level, level 5 and 25 reached around 1/4;

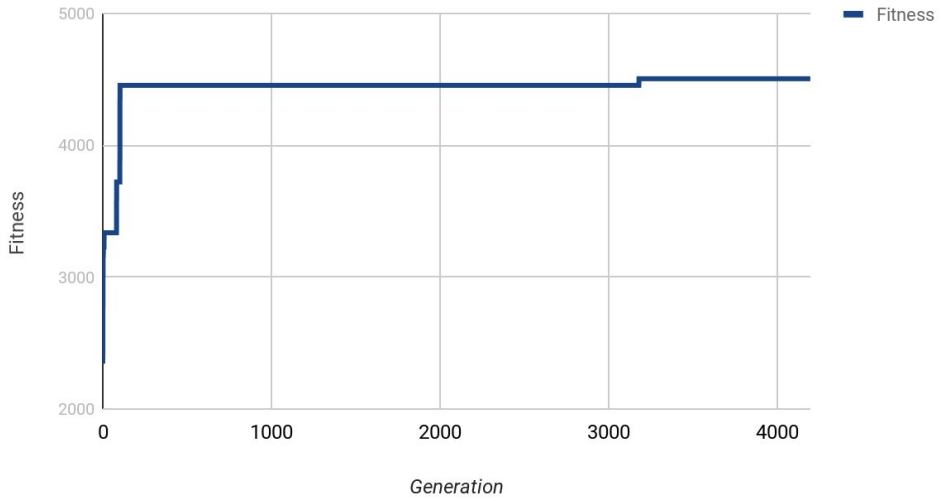
Two Point Crossover, Level 0, Mutation Parameter = 3



Two Point Crossover, Level 5, Mutation Parameter = 3



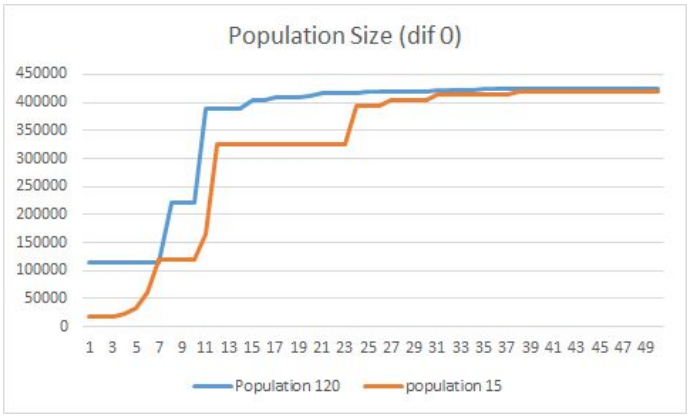
Two Point Crossover, Level 25, Mutation Parameter = 3

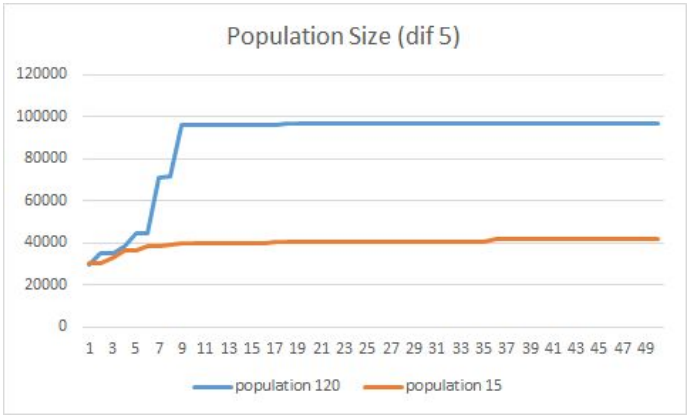


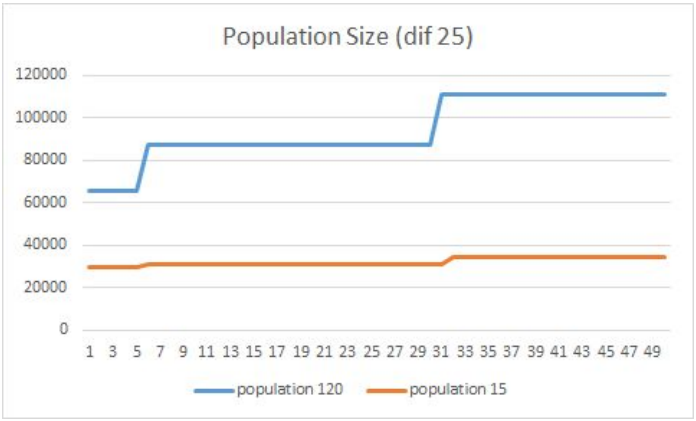
Second-Time Data Collection

Population Size

We set the population size to two different values (15 & 120) to run on three difficulties. The evolving graphs are shown below:



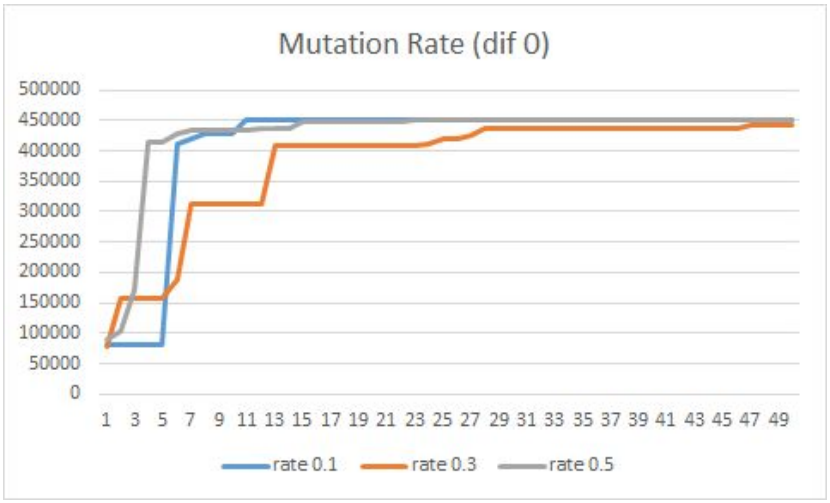


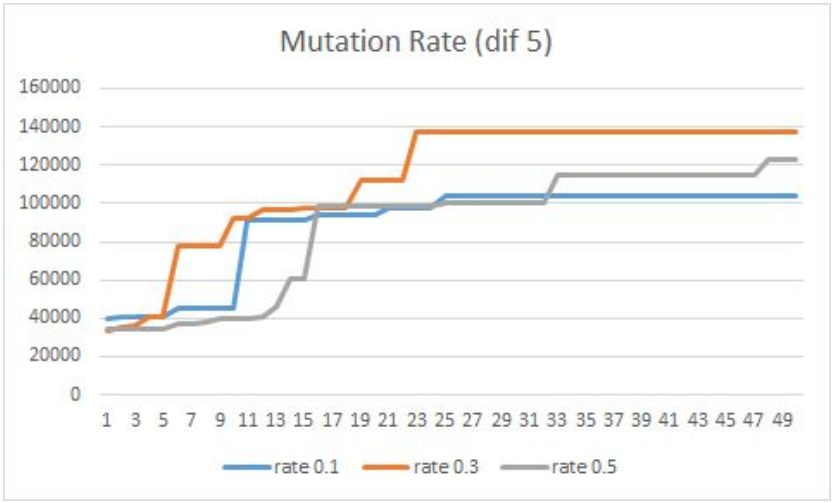


It is obvious to see that population size has a big impact on the result, that larger population size results in a significantly better agent and evolving speed.

Mutation Rate

We set the mutation rates to three different values (0.1, 0.3, 0.5) and train the agent on two difficulties (1, 5). The results are shown below:

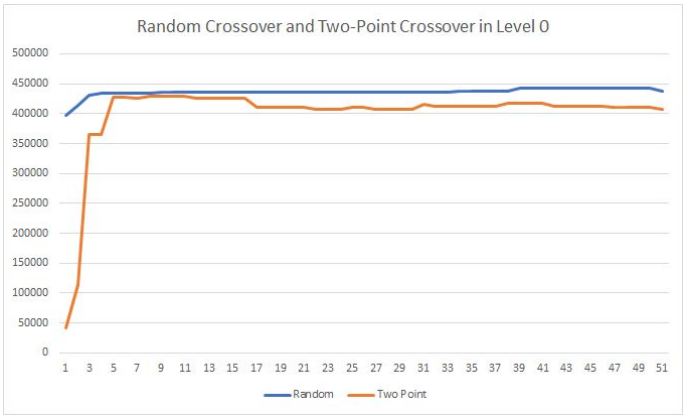


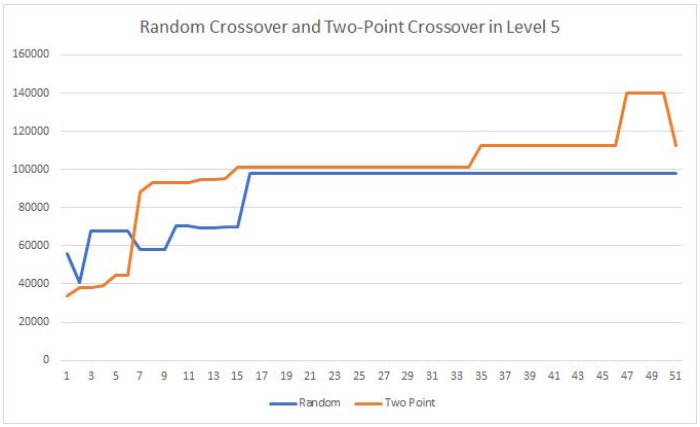


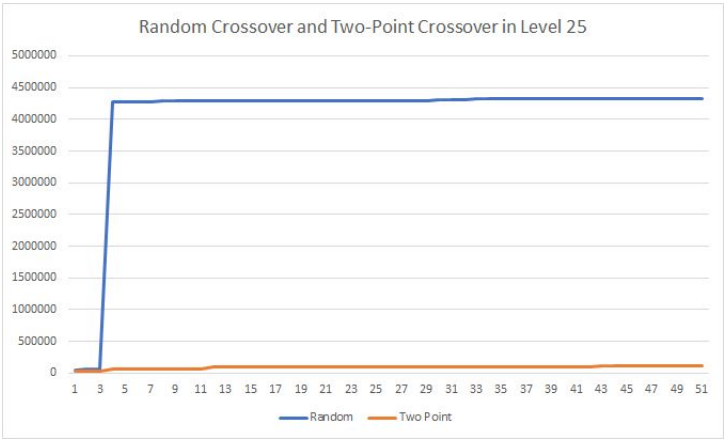
In the evolving process of difficulty 0 levels, the three mutation values do not have significantly different results.
However, in the evolving process of difficulty 5 levels, the mutation value of 0.3 have a slightly better result than the other two.

Two Point Crossover & Random Crossover

We tested two crossover strategies, the two point crossover and random crossover, on three different difficulties (0, 5, 25). The results are shown below:



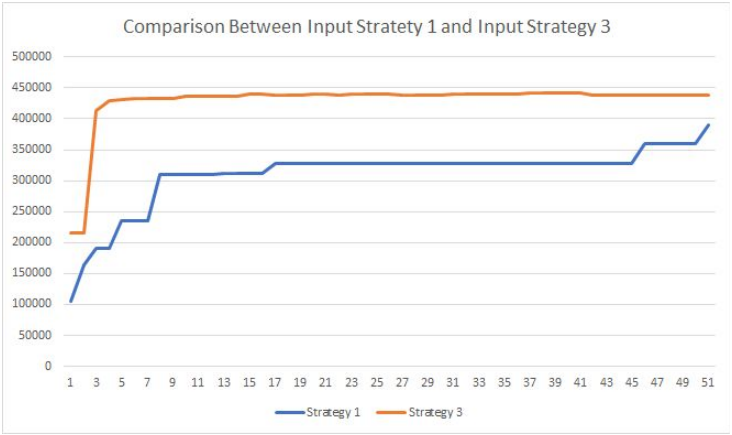


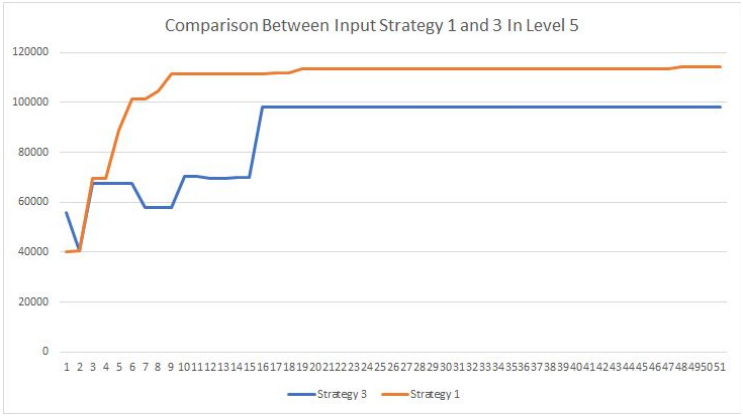


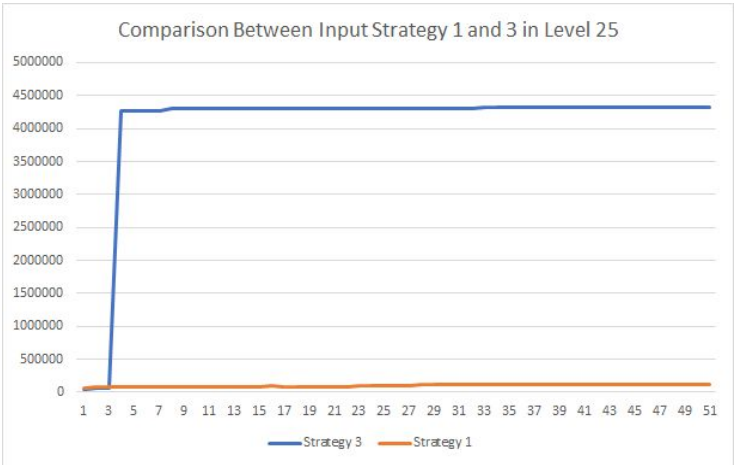
The resulting graphs do not show a clear advantage of using either of the two strategies. The large jumped showed in the third graph is because the agent learnt a way to repeatedly stomping on a shell. Therefore the big difference in the graph cannot show the advantage of either method.

Input Strategy

We run each level with input strategy 1 and 3, and below is the result observed from the first 50 generations.



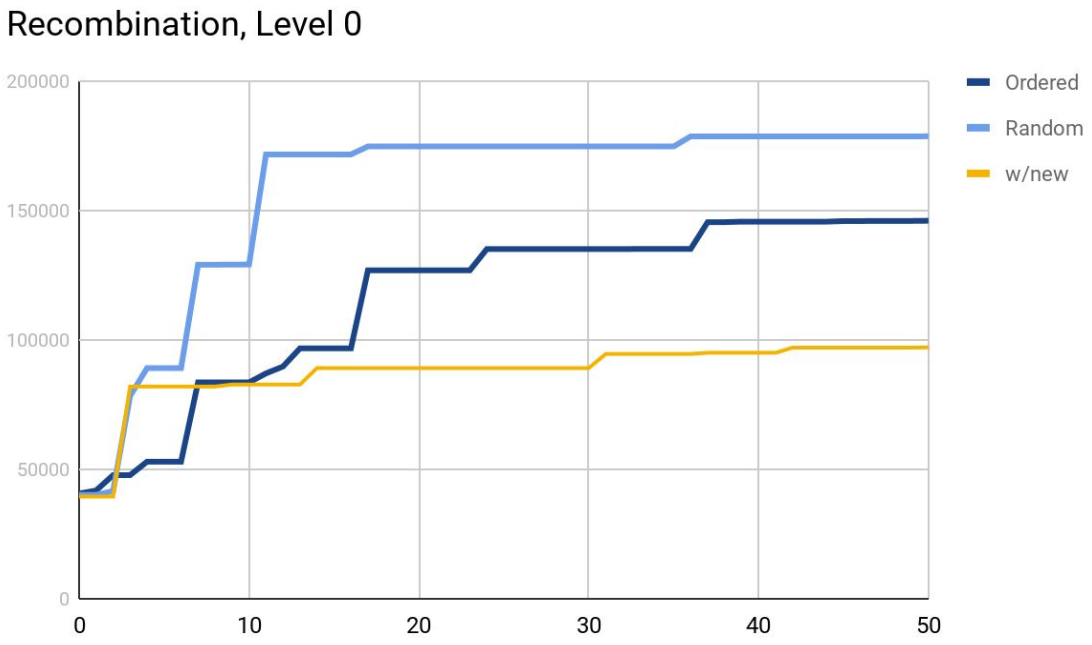


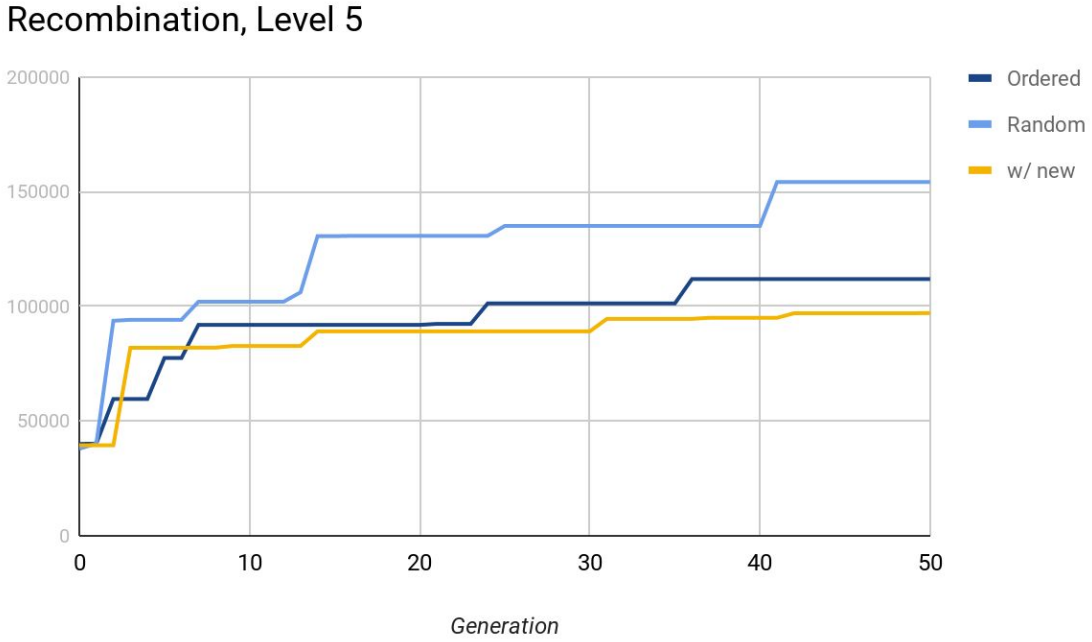


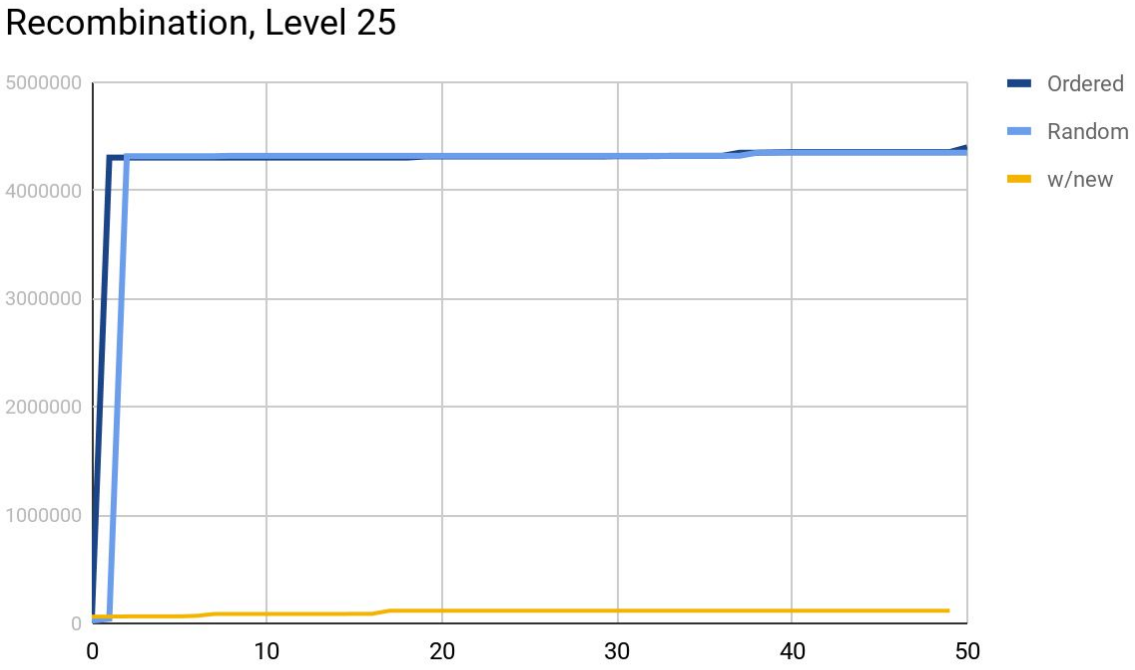
As shown in the graphs, with more processed information, the agent is able to grow faster in the first 50 generations.

Recombination Strategy

Below is the graphs of running 3 levels (difficulty 0, 5, 25) with each recombination strategy, which the first two choose parents without new random weights.







Random recombination shows some advantages here. In the last level, the randomness somewhat prevents the agent from stuck in the “score boosting” play style.

Conclusion

After performing numerous experiments using our NN and ES module, we understand several properties of this structure. Our result shows that larger population evolves more efficiently in the beginning. As the evolution process continues, an appropriate mutation rate is necessary in order for the agent to get out of the local optimal. The two crossover strategies have not yet shown obvious difference during the first 50 generations. Processed information is easier for the NN to evolve.

We also found that there was still a lot space to explore. After trying different strategies, one of our agents is able to complete the level of difficulty 0 with default random seed (see screenshot below). However, the agent has overfitting problems that causing it to not perform well or evolve efficiently in other levels or difficulties. Also there are chances that the agent reaches the local optimal by repeatedly stomping on objects and boost its fitness, causing huge bumps in the graph. Due to time limitation, we couldn't evolve our agents for more generations and generate more graphical results. NN+ES has a lot of rooms to explore, and the potential result can be very exciting.