**Game of Life with Genetic Algorithm**

Ben Zhang 001305466

Ci Chen 001061318

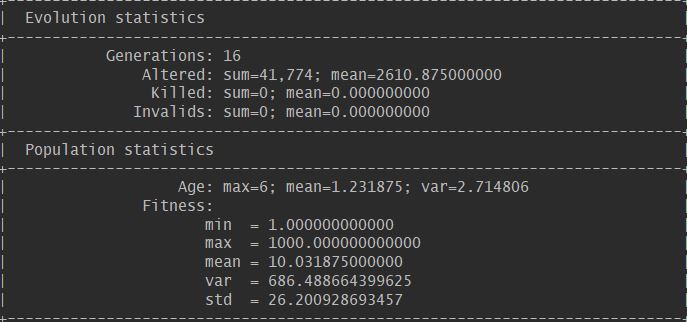
Bingxi Zhuang 001306651

**Introduction**

The main problem of this project is to find out the starting pattern of game of life, which is proven to grow over time and generate the most generations among all populations.We need to use Genetic Algorithms to solve this problem. Genetic Algorithms use the same techniques as does nature in order to find a good solution for a problem. In this project we used Jenetics to implement genetic problems.

**Observation**

Our goal is to use the genetic algorithm to find the optimal starting pattern. This starting pattern can make a group of cells live more than 1000 generations in Game of Life games. Actually, it is easy to find those start patterns by genetic algorithms. For example, we can get a result after about 16 generations of evolutions without any optimizations:



Here is the result when population size is 1000, the probability of crossover is 0.2, number of maximum evolution is 1000.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| probability of mutation | generation number | | | |  |  |  |  |  |  | average generations |
|  | No. 1 | No. 2 | No. 3 | No. 4 | No. 5 | No. 6 | No. 7 | No. 8 | No. 9 | No. 10 |  |
| 0.1 | 7 | 8 | 1 | 6 | 5 | 18 | 41 | 7 | 4 | 4 | 10.1 |
| 0.2 | 4 | 3 | 3 | 9 | 6 | 1 | 5 | 2 | 6 | 4 | 4.3 |
| 0.3 | 6 | 3 | 8 | 14 | 11 | 4 | 8 | 5 | 3 | 12 | 7.4 |
| 0.4 | 6 | 25 | 8 | 1 | 17 | 14 | 13 | 2 | 11 | 6 | 10.3 |
| 0.5 | 9 | 14 | 4 | 45 | 24 | 10 | 6 | 5 | 2 | 9 | 12.8 |

According to this graph, when mutation probability is 0.2, the generations is more stable and the average generation is the least among the five.

Here is the result when population size is 1000, mutation probability is 0.2 and the number of maximum evolution is 1000.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| probability of crossover | generation number | | | |  |  |  |  |  |  | average generations |
|  | No. 1 | No. 2 | No. 3 | No. 4 | No. 5 | No. 6 | No. 7 | No. 8 | No. 9 | No. 10 |  |
| 0.1 | 11 | 40 | 26 | 27 | 9 | 2 | 30 | 8 | 11 | 7 | 17.1 |
| 0.2 | 8 | 14 | 7 | 11 | 5 | 25 | 12 | 1 | 24 | 3 | 11 |
| 0.3 | 2 | 3 | 10 | 4 | 7 | 2 | 3 | 2 | 14 | 12 | 5.9 |
| 0.4 | 1 | 7 | 12 | 8 | 5 | 6 | 4 | 5 | 4 | 7 | 5.9 |
| 0.5 | 7 | 3 | 8 | 8 | 5 | 2 | 8 | 5 | 3 | 4 | 5.3 |

According to this graph, when crossover probability is 0.5, the generations is more stable and the average generation is the least among the five.

We selected 3 types of survivorsSelector as candidate selector, which are TournamentSelector, TruncationSelector and Linear-rank Selector. The TournamentSelector selects best individual from a random sample of s individuals is chosen from the population Pg. In truncation selection individuals are sorted according to their fitness and only the n best individuals are selected. In linear-ranking selection the individuals are sorted according to their fitness values. The rank N is assigned to the best individual and the rank 1 to the worst individual. Here is the results when mutation probability is 0.2, crossover probability is 0.5.

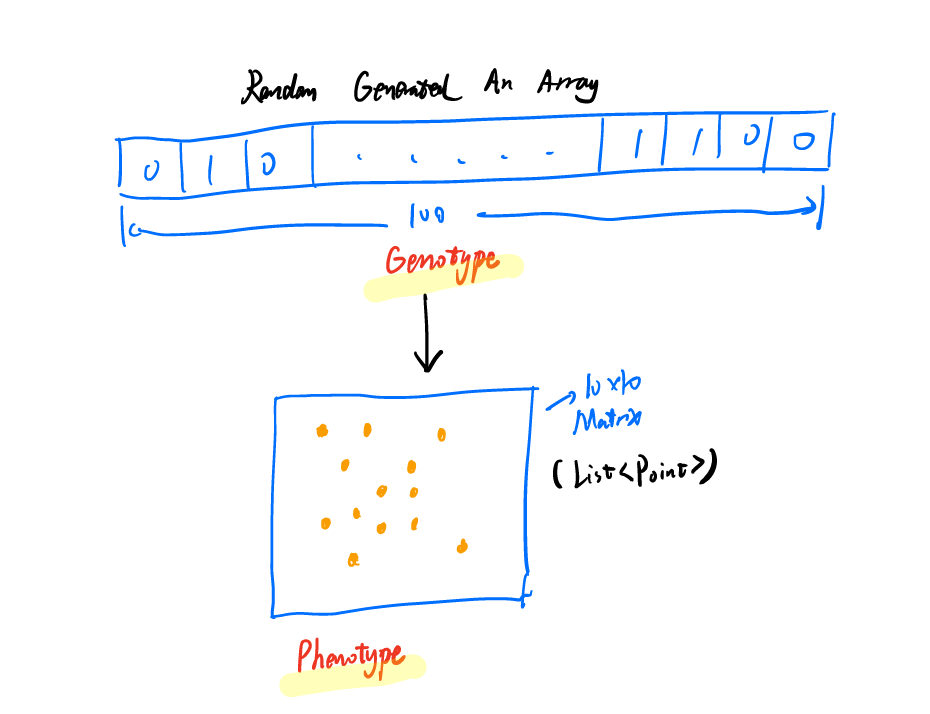
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| type of selector | generation number | | | |  |  |  |  |  |  | average generations |
|  | No. 1 | No. 2 | No. 3 | No. 4 | No. 5 | No. 6 | No. 7 | No. 8 | No. 9 | No. 10 |  |
| TournamentSelector | 7 | 3 | 8 | 8 | 5 | 2 | 8 | 5 | 3 | 4 | 5.3 |
| TruncationSelector | 16 | 2 | 2 | 2 | 6 | 9 | 9 | 3 | 4 | 3 | 5.6 |
| Linear-rank Selector | 13 | 8 | 159 | 17 | 5 | 6 | 10 | 11 | 1 | 3 | 23.3 |

As we can see, tournamentSelector has the least average generation, so we choose TournamentSelector as the selector.

**Design**

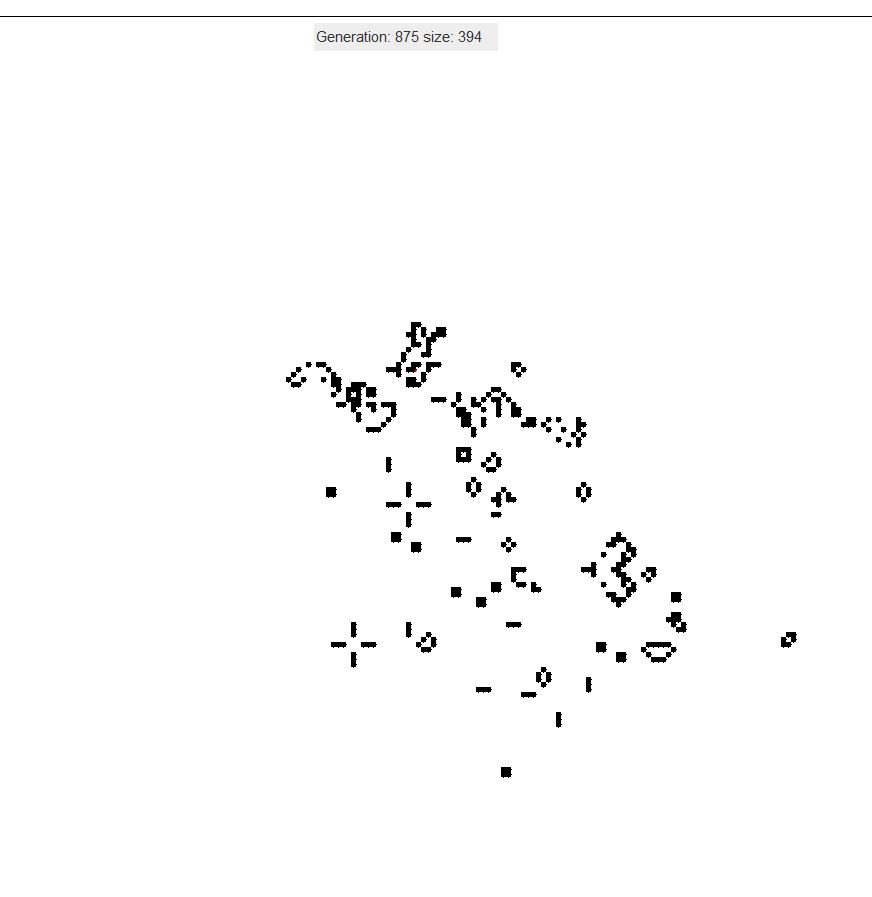
Firstly, we make the game of life problem convert to a problem which can be finished by genetics algorithms. We define the genotype, phenotype and fitness function of this problem.

Before starting genetic algorithms, the program randomly generates a vector with size of 100, and the value of every element is 0 or 1. We define this vector as genotype in our project. Every genotype in our program has only one chromosome with the length of 100. The phenotype of our problem is a list of points derived from genotype. We iterate the genotype ( an int array with the size of 100). And if the value of the element is 1, it means a live cell in the grid.We use a simple mathematical method to get the position and initialize a point with the position. Then after the iteration, we get a list of points. This is our genotype.



Fitness function is essential to a GA problem. In our project, we directly define the generation of the Game of Life last as the fitness. Because we want to get the pattern who can last longer. So the generation of the game lasts is the fitness of every individual in this system.

There is a group of cells in the generation of 875 in the game(in our GUI ), and the count of points reached 394.It is only about 50 at the beginning. That means it has a strong fitness:



Then we chose the mutator and crossover of the program. We didn’t implement the mutator because Jenetics already has one. According to our observation, we think the best probability of the mutator is 0.1 or 0.2. Mutator is to increase the diversity of the population.

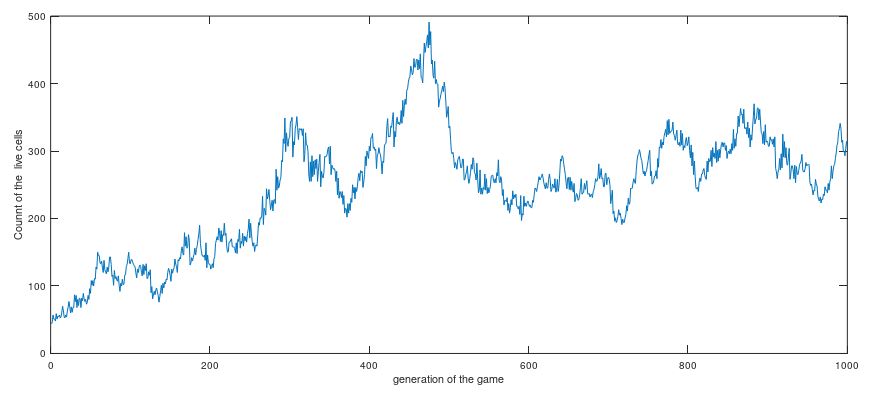
Crossover is significant in GA problems. We can use crossover to get the offspring of the population. We mainly test two kinds of wide-used crossover: singlepoint and uniform. Singlepoint crossover is a classical form of crossover. Yet it may produce slower than the uniform crossover. But we found that these two kinds of crossovers performs both well in our problem. We think both of them are okay with game of life.

We use survivor selector and offspring selector in our problems. We use Tournament selector as our survivor selector because it can choose the best individuals randomly in the population. We defines that only selector the top 10% individuals from the survivors. Roulette-wheel selector is fitness proportional without sort individuals in the population. We use this to select the offsprings.

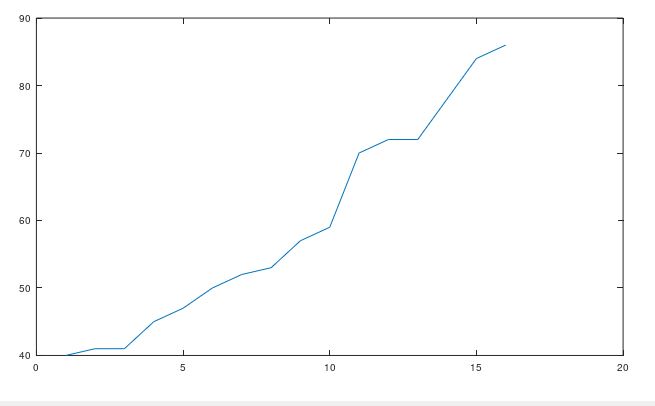
It is obvious that if the population size is bigger, the algorithms will have better results. However, we don’t need very large population because we have mutation and crossover. In our problem, we set the population size is 100.

**Result**

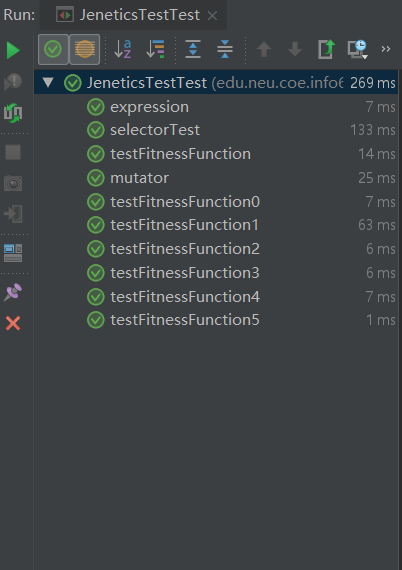
There is a graph of one game which lasts for 1000 generations. Its x-axis is the generation of the games, and the y-axis is the count of the live cells in the group. The count of cells at the beginning is about 50. It can reach about 500 in the game:



However, we found that if we ask the game must grow by generations(count bigger than the previous game’s ), the game can’t last very long:



**Test**



We test our expression of genotype and phenotype, mutator, selector and fitness. They all pass.

**Conclusion**

According to the analysis above, we used Genetic Algorithms to find better starting pattern, which generated 1000 generations by game of life rules. Genetic Algorithm simulates the process of natural selection and evolution in the real world. It is a bright algorithms and Jenetics helps us a lot. To sum up, GA is really a useful tool for finding the best solution.