Game of Life with GA

1. Objective

1.1. Problem Description

Using Genetic Algorithm (GA) to find a pattern that lives and grows forever in Conway's Game of Life and show the result with UI (UI is only required for three-person team).

We can use Jenetics or other things online or write our own GA. In our case, we wrote our own GA.

1.2. Genetic Algorithm

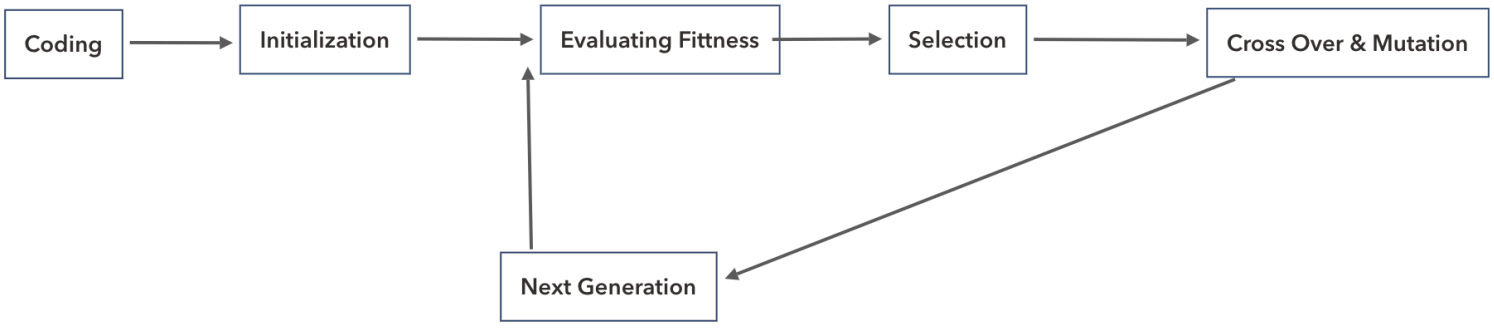
I. **Definition**: A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. (Wikipedia)

II. **Advantages**: Directly perform operations on structural objects. There are no restrictions on differentiation and function continuity. It has inherent implicit parallelism and better global optimization capabilities. It uses probability-optimizing search methods. Without the need to determine the rules, it automatically obtain and guide the optimized search space, and adaptively adjust the search direction.

III. **Disadvantages:** It's not guaranteed to find the best solution. It may end up with a good solution.

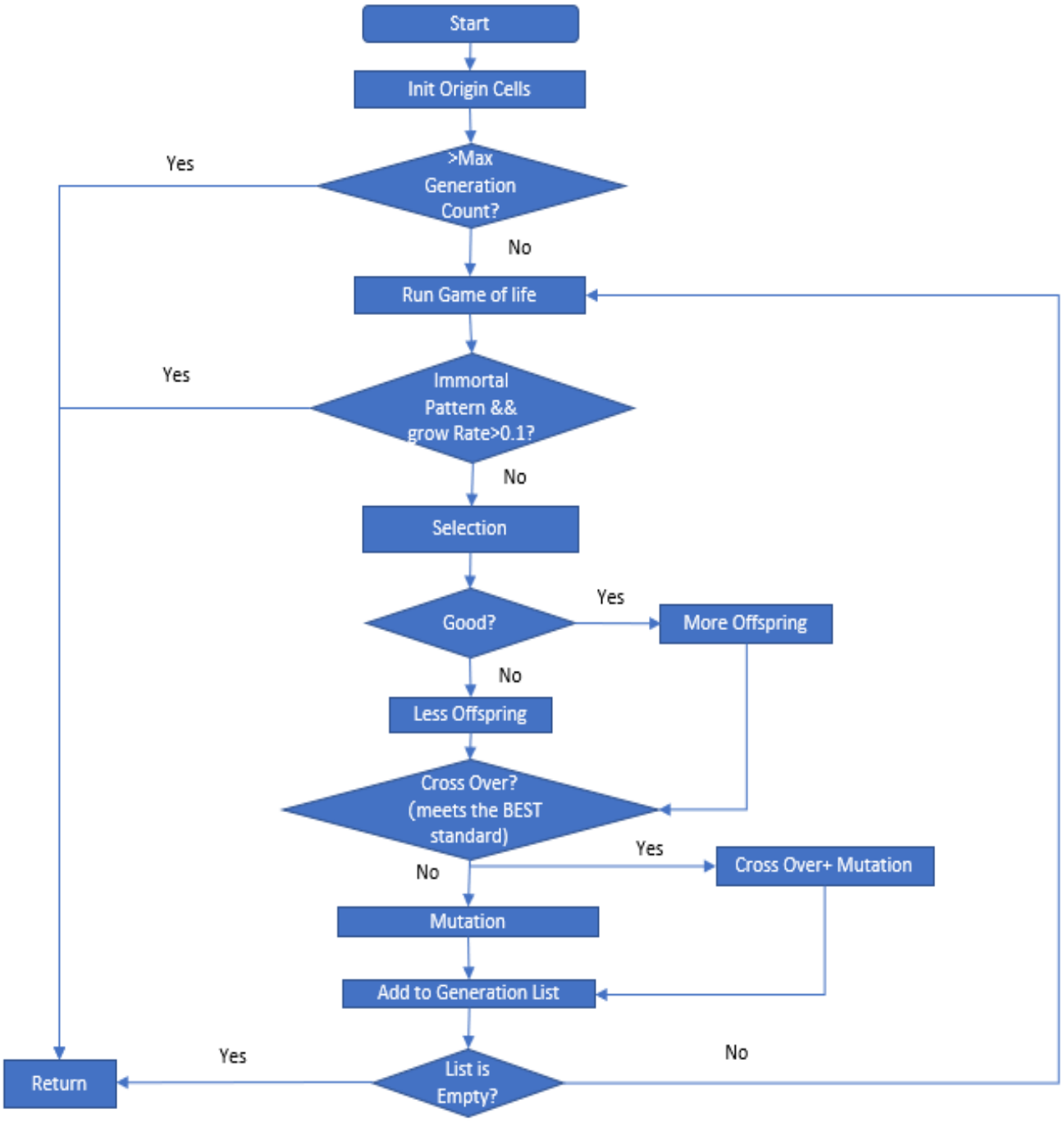
IV. **Structure:** A GA usually consists of following 5 parts: **coding**, **initialization**, **selection**, **genetic operators** and **termination**. (Implementation of these parts in this program is in Sec 2.2.)

The relationship between the beginning four parts is shown in the following picture, in which 'Cross Over & Mutation' represents genetic operators.



2. Program

2.1. Program Structure

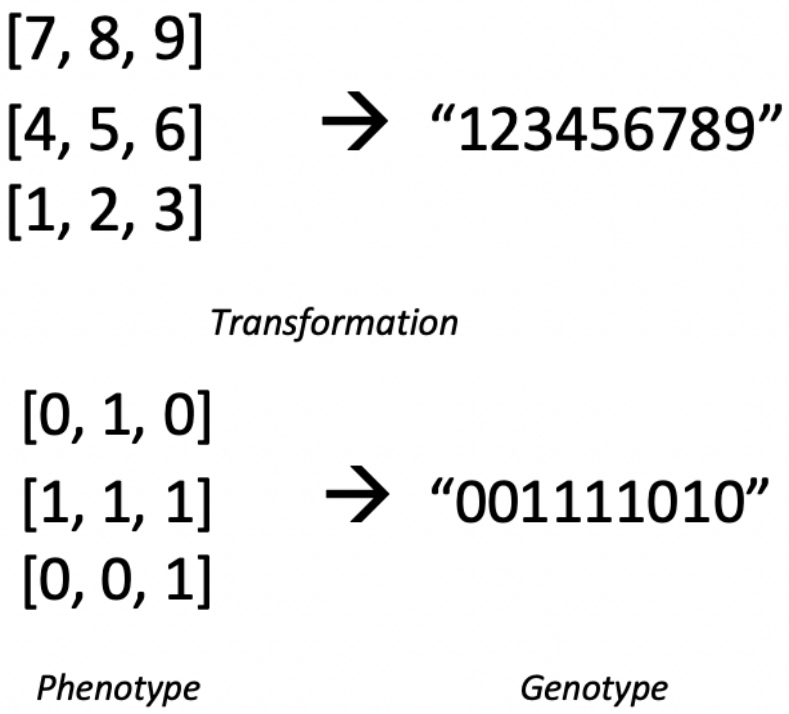


2.2. Implementation of GA

2.2.1 Coding

Coding is representing phenotype in a computer-understandable method, in other word, transferring phenotype into genotype.

In this program, phenotype is a pattern that shows in a binary array. We chose to use 01 string as its genotype. The order is shown in pic. *Transformation* and followed with an example.



2.2.2 Initialization

Population Size

Population size is depended on the size of the binary array of first generation. We chose 2DArray.length as the size, which in this case is 15, but you can change it by changing the size of initial 2Darray.

Random

By using method Math.random(), each cell has 50% chance of being alive.

2.2.3 Fitness Function

Fitness

Fitness is the score of each individual after running Conway's Game of Life. In this case, we use 'generation' and "grow rate" as the fitness. If the individual isn't immortal, the bigger the generation, the better and it's growth rate should be bigger than 0.1.

2.2.4 Selection Function

The purpose of this step is to optimize next generation by using the fittest genes and eliminates the least fit genes in this generation.

The algorithm of this part is regarding its generation in game of life return the size of its offspring. And for extra outstanding individuals, they are allowed to cross over and mutate. While others can only mutate.

Following is the numbers of offspring regarding different generations in this case.

generation	offspringSize	CrossOver
[0,3)	0	NO
[3,5)	2	NO
[5,10)	4	NO
[10,20)	6	NO
[20,50)	10	NO
[50,100)	16	NO
[100,500)	26	NO
[500,800)	42	YES
[800,1200)	68	YES
[1200,1500)	110	YES
[1500,1800)	178	YES
[1800,	288	YES

For the ones that are immortal, if its growth rate is larger than 0.1, then we found the best solution. But if not, they need to go to next selection. If growth rate>-0.3, it has the ability to cross over otherwise only mutation.

2.2.5 Cross Over & Mutation

Cross Over

For individuals who meets the requirement for cross over, they become father or mother if father or mother position is empty. As soon as father and mother both filled, these two can cross over and produce the average amount of children regarding their generation.

The length of cross over: a random num from 0 to half the length of parent gene.
Starting position: a random number from 0 to parent.length/2

Mutation

For a giving probability (0.9 in this case), generates a double dou from Math.random(), if dou<(1-probability) then mutate.
Mutate: flip the value in the position. E.g. 0->1, 1->0

2.3. Unit Tests

2.3.1 Initialization Test

```
Init Cells Test:
0 0 0
1 0 1
1 1 1
---------------------
1 1 0
0 0 0
0 1 0
---------------------
1 1 1
0 1 0
0 0 0
---------------------
0 0 1
0 1 0
0 1 1
---------------------
1 0 0
1 1 0
0 0 1
---------------------
---------------------
Init GenTypeCells List Test:
[110000100, 001100100, 001111000, 001001001, 111011111]
```

2.3.2 Coding & Decoding Test (phenotype->genotype & genotype -> phenotype)

```
Transfer form Genoty to phenotype Test:
Genotype: 1 1, 0 1, 2 1
Phenotype:
0 1 0
0 1 0
0 1 0

----------------------------------------
Transfer form Phenotype to GenotyTest:
0 1, 1 1, 2 1
--------------------------------------------------
Transfer small arry to larger array Test:
Small array:
1 1
1 1
---------------------
Large array:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 1 0 0
0 0 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

2.3.3 Selection Test

```
Selection Method Test:
0 == 0?
------------------
6 == 6?
------------------
10 == 10?
------------------
26 == 26?
------------------
```

2.3.4 Mutation Test

```
Mutation Test:
Before Mutation:
0 1 0
0 1 0
0 1 0
------------------------
After mutatoin:
0 1 0
1 1 0
0 0 0
```

2.3.5 Cross Over Test

```
Corssover Test:
000110001 1010111000 crossover length: 4
Corssover Result: [000111101, 001110001, 100110001, 000111001]
```

3. Result

Here's the pattern that we found immortal in Game of life with a growth rate that is larger than 0.1.

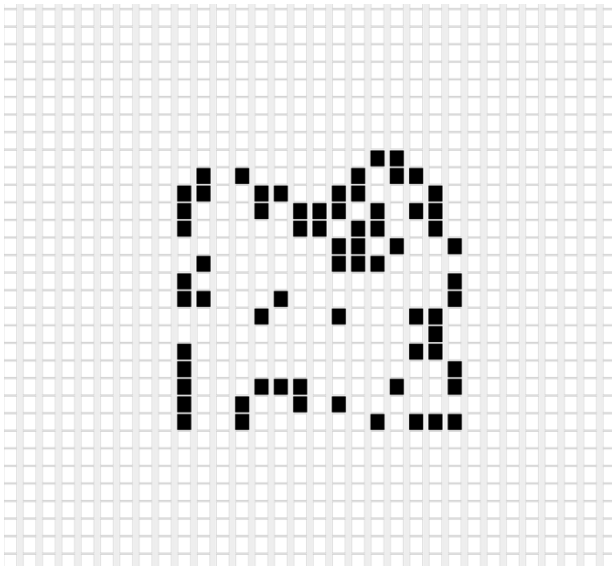
```
generation 704; grid=Grid{generation=704, groups=[generation 704, origin = {-13,
    [{0, 0}, {0, 1}, {-2, 0}, {-2, 1}, {-1, -1}, {-1, 2}, {-17, 2}, {-17, 8}, {-1
generation 704;
count=190
Group generation: 704
Terminating due to: having matching previous games
growthRate=0.10354609929078014
00110001011001111011101111011001010100110111010111101011101101101100001100000001
2 0, 3 0, 7 0, 9 0, 10 0, 13 0, 14 0, 0 1, 1 1, 3 1, 4 1, 5 1, 7 1, 8 1, 9 1, 10
```

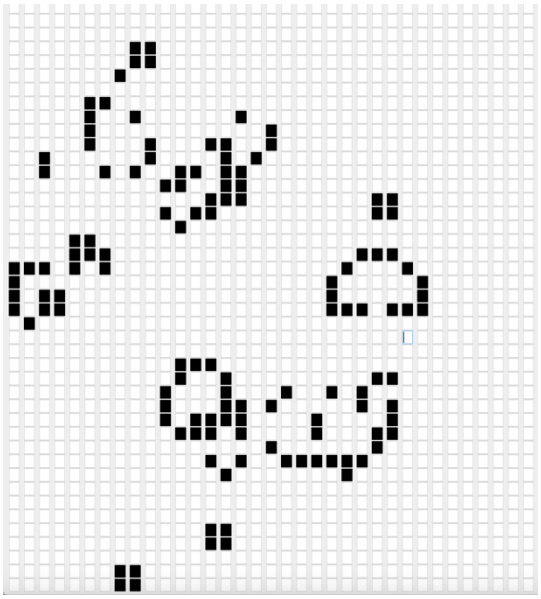
Seed:
00110001011001111011101111011001010100110111010111101011101101101100001100000000110100100000101011111111010010010010111000001011010101101100001101100101011000101100111100101100101101010111100110010101011010101110000101100001101

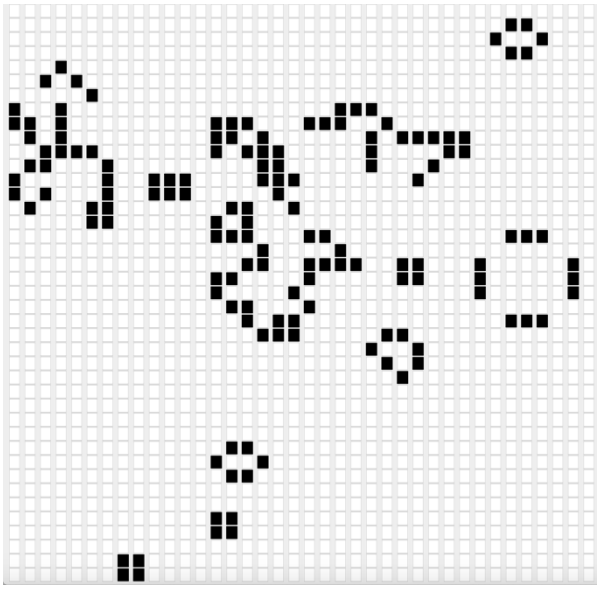
Coordinate: 2 0, 3 0, 7 0, 9 0, 10 0, 13 0, 14 0, 0 1, 1 1, 3 1, 4 1, 5 1, 7 1, 8 1, 9 1, 10 1, 12 1, 13 1, 1 2, 3 2, 5 2, 8 2, 9 2, 11 2, 12 2, 13 2, 0 3, 2 3, 3 3, 4 3, 5 3, 7 3, 9 3, 10 3, 11 3, 13 3, 14 3, 1 4, 2 4, 4 4, 5 4, 10 4, 11 4, 4 5, 5 5, 7 5, 10 5, 1 6, 3 6, 5 6, 6 6, 7 6, 8 6, 9 6, 10 6, 11 6, 12 6, 14 6, 2 7, 5 7, 8 7, 10 7, 11 7, 12 7, 3 8, 5 8, 6 8, 8 8, 10 8, 12 8, 13 8, 0 9, 1 9, 6 9, 7 9, 9 9, 10 9, 13 9, 0 10, 2 10, 3 10, 7 10, 9 10, 10 10, 13 10, 14 10, 0 11, 1 11, 4 11, 6 11, 7 11, 10 11, 12 11, 13 11, 0 12, 2 12, 4 12, 5 12, 6 12, 7 12, 10 12, 11 12, 14 12, 1 13, 3 13, 5 13, 6 13, 8 13, 10 13, 12 13, 13 13, 14 13, 4 14, 6 14, 7 14, 12 14, 13 14

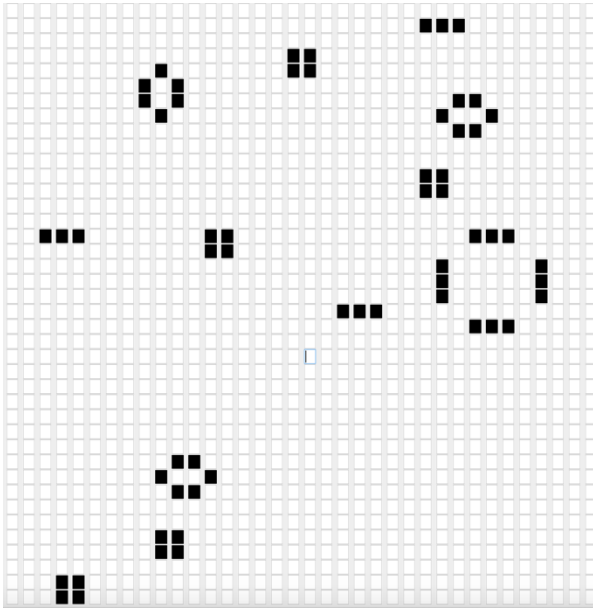
Growth Rate: 0.1035

Screenshots of UI:









4. Conclusion

We can see from UI that this pattern ends up with a fixed count of living cells instead of growing. However, the growth rate is actually larger than 0.1. The explanation to this confliction is as followed.

I. The given method of calculating growth rate is (i-j)/n, where i is the count of living cells in current generation, j is that of the very first generation and n is the number of generations. While the actual definition of growing is after each cycle the number of living cells should be growing. Current counting method does not calculate the cycle but the whole process.

   Without finding the right method to calculate growth rate, in other words, the right fitness method, selection cannot be efficient. As a result, the odds of finding growing immortal pattern is small.

II. Our computer is not good enough. Break down happens often.

III. Time is limited. We didn't find the pattern after running the program all night.