Game Of Life

——Use genetic algorithm to find the fittest individual and its offspring

INFO6205_2019Fall
Team Number: 107

AUTHOR:

INFO6205_2019Fall
Team Number: 107
Team Member:
Jinnuo Che
Xing Tong
Yu Ren

Content

1 INTRODUCTION

1.1 Conway's Life Game[1]

Conway's life game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves. The game is a cellular automation which imposes fixed rules on the cells playing on an infinite two-dimensional discrete grid.

1.2 Genetics Algorithm

In the Conway's life game, its evolution is determined by its initial state. So finding an optimal initial state is crucial for the game's result. However, we cannot expect an optimal initial state to produce the same good result each time and we cannot ensure the result is always the best.Thus the optimal initial state is random, uncertain and diverse.

And the Genetics Algorithms are non-deterministic and have no analytical solution that just meets the requirements of our problem - finding an optimal initial state for Conway's life game. Therefore, genetics algorithm is an effective method for our problem.

1.2.1 First glance of Genetics Algorithm

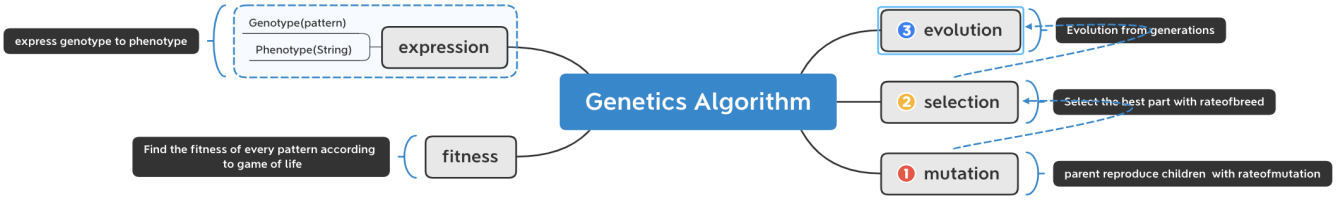

Figure 1. Structure of Genetics Algorithm

1.2.2 Properties about Genetics Algorithm[2]

• GAs use the same techniques as does nature in order to find a good (not necessarily the best) solution for a problem.
• GAs are non-deterministic: you do not expect them to give you the same solution each time you run.
• GAs are useful for problems that have no analytical solution, especially problems with are NP.
• GAs are well-suited to problems with huge solution spaces

•

2 PROBLEM DESCRIPTION

2.1 Overview

In the Conway's life game, its evolution is determined by its initial state. So finding an optimal initial state is crucial for the game's result. Genetic algorithm is non-deterministic and have no analytical solution. And it can simulate the natural evolution, mutation and selection,etc. Therefore, we introduce the Genetic Algorithm to get better from the better so that we can find good starting pattern.

2.2 Concepts of Genetics Algorithm

2.2.1 Expression

Expression is the "mapping" of genotype to phenotype. In an individual gene pattern, the expression can obtain the sequence of gene from Genotype and then transfer them to an individual as Phenotype;

2.2.2 Genotype

The set of replicable and heritable information which is a property of a candidate solution ("organism")— the genes;[3]

2.2.3 Phenotype

The traits of the candidate which affect how good a solution it **is;**[4]

2.2.4 Mutation

Mutation is a way for parents to reproduce the children. In the natural world, genotypes can be copied from parents to child in a certain ratio.

2.2.5 Selection

Selection in Genetics Algorithm is a method to simulate the natural world selection period - natural selection, survival of the fittest. Selection make sure the good gene can be inherited and the bad gene will be eliminated.

2.2.6 Fitness

Fitness is a parameter that can evaluate how well the candidate solution solves the problem. In the natural world, Fitness can imitate the adaptability of every individual.

3 PARAMETER DESIGN

3.1 initialpopulation

The total number of initial population.

3.2 maxgeneration

The maximum number of generations.

3.3 points

The number of points individually.

3.4 range

The range of points (x and y).

3.5 rateofbreed

Proportion of organisms that survive and breed. It determines how many individuals can survive after one selection.

3.6 rateofmutation

Proportion of mutation. The reteofmutation can influence the speed of the mutation.

4 PROGRAM STRUCTURE

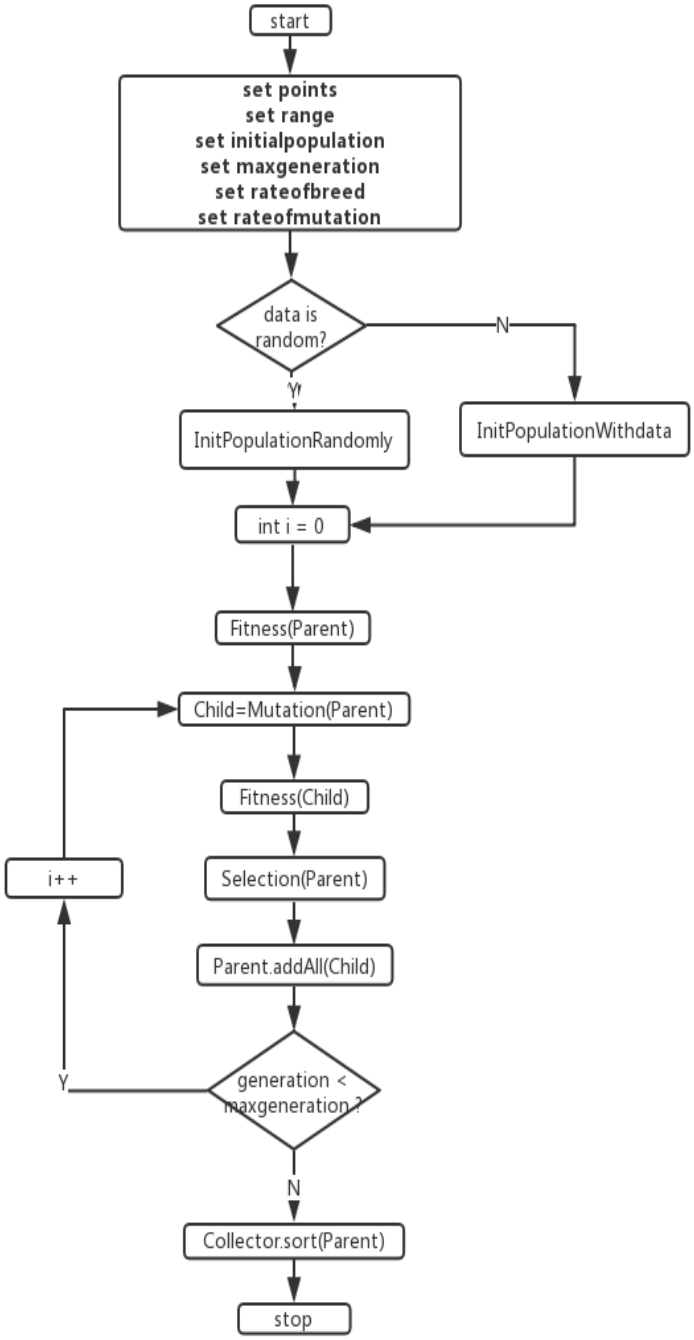

Figure2. Process of Program

•

5 PROGRAM DESIGN

5.1 Calculation algorithm of the fitness

Though mutation could bring new features to the species, we can not guarantee the new feature that will increase the survival rate of the whole group. Because mutation is random. But nature will select the fittest one to survive and kill the one who is least responsive to change. In our project, we define the fitness to describe the species' ability to respond to the natural changes.

```
public static void Fitness(List<Pattern> pop)
  // Get the generation though Game
  for(Pattern pattern : pop){
      Game.Behavior generation = Game.run(0L, Expression(pattern));
      pattern.setFitness(generation.generation);}
```

For every input pattern, we will put them into Game method(Game method is to create the generation). After running Game method, we can get the number of the generation this pattern generates. And this number reflects the fitness.

5.2 Mutation algorithm

In our project, we design the mutation to reproduce next generation instead of the crossover. Usually, mutation only happens in DNA crossover. Because in the experiment, we set the limited generations, so we can not go infinitely as nature. So, to simplify the problem, we choose mutation to replace crossover. In our algorithm, we set the principle to mutate. First, we set the rate of mutation. We will operate separately to different parts. For the first ⅛ (for example, mutation rate is 0.5 and multiples with ¼) of points, we will change their x integer. And the second ⅛ of points, we will change their y integer. And the last ¼ of points, we will change both x and y. Because we use a random number to judge whether this pattern will mutate every time, the whole process of mutation is random.

```
if(randomnum < 1000 * rateofmutation / 4){
  int Changedpoint = random.nextInt(points);
  int x;
  while(true){
      Boolean x_plusminus = random.nextBoolean();
      int x_value = random.nextInt(range);  // the value
      x = x_plusminus? x_value : 0 - x_value;
      Point point = new Point(x, pattern.getY(Changedpoint));
      if(!pattern.getSetOfpoint().contains(point)) break;}
  Pattern newpattern = new Pattern(pattern);
  newpattern.setX(Changedpoint, x);
  newpattern.resetSetOfpoint();
  popAfterbreed.add(newpattern);}
```

•

```
if(randomnum >= 1000 * rateofmutation / 4 && randomnum < 1000 * rateofmutation / 2){
  int Changedpoint = random.nextInt(points);
  int y;
  while(true){
     Boolean y_plusminus = random.nextBoolean();
     int y_value = random.nextInt(range);
     y = y_plusminus? y_value : 0 - y_value;
     Point point = new Point(pattern.getX(Changedpoint), y);
     if(!pattern.getSetOfpoint().contains(point)) break;}
  Pattern newpattern = new Pattern(pattern);
  newpattern.setY(Changedpoint, y);
  newpattern.resetSetOfpoint();
  popAfterbreed.add(newpattern);}

// Change x and y of point
if(randomnum >= 1000 * rateofmutation / 2 && randomnum < 1000 * rateofmutation){
  int Changedpoint = random.nextInt(points);
  int x;
  int y;
  while(true){
     Boolean x_plusminus = random.nextBoolean();
     int x_value = random.nextInt(range);  // the value
     x = x_plusminus? x_value : 0 - x_value;
     Boolean y_plusminus = random.nextBoolean();
     int y_value = random.nextInt(range);  // the value
     y = y_plusminus? y_value : 0 - y_value;
     Point point = new Point(x, y);
     if(!pattern.getSetOfpoint().contains(point)) break; }
  Pattern newpattern = new Pattern(pattern);
  newpattern.setX(Changedpoint, x);
  newpattern.setY(Changedpoint, y);
  newpattern.resetSetOfpoint();
  popAfterbreed.add(newpattern);}
```

5.3 Selection algorithm

In the program, we put the pattern back to the game and figure out how many generations it can run. We use this result to describe the fitness. In the selection process, we implement compareTo() by comparing their fitness so that we can rank the patterns.

```
public int compareTo(Pattern pattern) {
```

•

```
    return (int) (this.getFitness() - pattern.getFitness());
}
```

Based on the rank, we use selection to kill part of the species (Ratio is set as rateofbreed). So, after selection, the species only with better gene will survive. In consequence, the species will improve their fitness after generations.

```
public static void Selection(List<Pattern> pop)
    Collections.sort(pop);
    int len = (int) (pop.size() * rateofbreed);
    for(int i = 0; i < len; i++){
        pop.remove(0);}
```

5.4 Expression algorithm

In nature, biologists study gene of different species and try to find out the secret of between genotype and phenotype. But in our two dimension world, we just easily define Genotype by points, and each point is shown on the grid with x and y. Meanwhile, each pattern has different fitness. We added it as a character of pattern. To avoid that two different cells are located in the same point of grid, we use hashset to guarantee enough points.

```
Comparable<Pattern>{
  private int[] x = new int[GA.points];
  private int[] y = new int[GA.points];
  private long fitness;
  private HashSet<Point> setOfpoint;
  public Pattern(){
    super();
    setOfpoint = new HashSet<>();}
```

And we write the specific method to transform the genotype to phenotype. We call this method expression.

```
public static String Expression(Pattern pattern){
  String result = "";
  for(int i = 0; i < GA.points; i++){
    if(i == GA.points - 1){
        result = result + pattern.getX(i) + " " + pattern.getY(i);}
    else{
        result = result + pattern.getX(i) + " " + pattern.getY(i) + ", ";}
  }
  return result;}
```

5.5 Evolution Algorithm

When species can mutate and select simultaneously, we call it evolution. Because selection method will avoid the bad mutation to affect the whole population. Maybe, mutation will produce extremely bad individual. But it will not survive in this round since there is a selection method. In our code, we leave two ways to input the initial pattern.

One way is random data. This is for the first several times to run our project, we do not have any original data. After running project several times, only top results will save in ”Goodpop_*.txt“ . We can use it as the seed value to run the second time. Each time we will save the top results. In this way, we can get the better population and reproduce the results.

```
public static List<Pattern> Evolution(){
    List<Pattern> initpop = new ArrayList<>(); //A list for initialising population
    InitPopulationRandomly(initpop);
//      InitPopulationWithdata(initpop, "Goodpop_10points.txt");  *******
    RecordInitialPop(initpop);
    Fitness(initpop);
    List<Pattern> Parent = initpop;
    List<Pattern> Child = null;
    for(int i = 0; i < GA.maxgeneration; i++){
        if(i == 0){
            Child = Mutation(Parent);
            Fitness(Child);
            Selection(Parent);}
        else{
            Parent.addAll(Child);
            Child = Mutation(Parent);
            Fitness(Child);
            Selection(Parent);}
        if(Parent.size() + Child.size() < GA.initialpopulation/10){
//              Parent.addAll(Child);
            break;}
    }
    Parent.addAll(Child);
    Collections.sort(Parent);
    RecordFianlPop(Parent);
    return Parent;}
```

5.6 Seed Value Captured

In our project, there are several files named “Goodpop_*.txt” to save the seed value that we can reproduce our results. These seed values were captured after several tests. We set the file name under certain rules, for example, “Goodpop_10points.txt” means we saved initial population of patterns that contain 10 points. If we want to reproduce the result, we need to replace InitPopulationRandomly(initpop); with InitPopulationWithdata(initpop, "Goodpop_10points.txt"); in Evolution function.

We collect each pattern in Comma-Separated Values(CSV) as a row. We analyze each row to produce a pattern so that we can initialize the population with previous data.

```
public static void InitPopulationWithdata(List<Pattern> initpop, String filename){
  String thisline = null;
  try(BufferedReader br = new BufferedReader(new FileReader(filename))){
      while((thisline = br.readLine()) != null){
          Pattern pattern = new Pattern();
          String[] data = thisline.split(":");
          String[] fields = data[1].split(",");
          for(int i = 0; i < fields.length; i++){
              String[] cmp = fields[i].split(" ");
              pattern.setX(i, Integer.parseInt(cmp[1]));
              pattern.setY(i, Integer.parseInt(cmp[2]));
      }
          pattern.resetSetOfpoint();
          initpop.add(pattern);
      }
  }
  catch (Exception e){
      e.printStackTrace();
  }
}
```

5.7 UI Design (Parallelly Show)

The main method of our project will get the best results. But we just capture best six patterns. and we put them into UI interface. In UI interface, we design the grids and points to visualize the result. To show six groups data at the same time, we build the threads to run them simultaneously. Clicking “run” button, the computer will compute for a period. After that there will be six JFrames showing on the screen. The pattern usually diffuses in the grid. If pattern crosses the border, though it will run continually, It will not show in the UI. Among our

•

experiment, we discovery that usually after 1,000 generations, the pattern will cross the border and UI will not update the visualization.

```
public static void main(String[] args){

    List<Pattern> result = Evolution();
    List<String> datas = new ArrayList<>();
    for(int i = result.size() - 1; i > result.size() - 7; i --){
        datas.add(" " + Expression(result.get(i)));
    }

    String title = "Game of life";
    int i = 0;
    for(String data : datas){
        JFrame frame = new JFrame();
        frame.setTitle(title);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//          frame.setLocationRelativeTo(null);
        frame.setLocation(500 * (i % 3), 500 * (i / 3));
        frame.setAlwaysOnTop(true);
        GameOfLife gol = new GameOfLife();
        frame.add(gol);
        frame.pack();
        frame.setVisible(true);

        gol.start(data);
        i++;
    }
  }
```

6 CHALLENGES IN CODING

6.1 Definition of Patterns

a)We represent the genotype as coordinate points(integer x and y) in the two-dimensional coordinate system.
b)Fitness is set as the generations that each individual can generate.
c)To prevent points duplication in a pattern, a Hashset<point> for each pattern was created to store current points.
d)In the selection method, we sort all the patterns according to their fitness. And in the pattern class, patterns with higher fitness have stronger survivability and will have better priority in the order.

6.2 Methods to Mutation

To ensure the diversity and randomness of mutations, points will be chosen randomly to mutate in rate of mutation , and there are three methods:
a)Only change axis “x”;
b)Only change axis “y”;
c)Change axis “x” and “y”.
We can change the rate of mutation to change the speed of mutation.

6.3 Maturity of Children

We assume that only after a generation, children can mature and then reproduce. To meet the requirement, we set two lists to store the Parent and Child separately. In each round, we put the Child from the last generation to the Parent and then allow them to reproduce a new generation that will be stored into Child list. Repeat the steps.

6.4 Shift of Origin of Coordinate System

In order to display the process of the game clearly, we shift the origin of the coordinate system from the initial place to the center place by operating the coordinates of points:

```
public void start(String datacsv)  {
    cGrid = new boolean[pixels.length];
    pGrid = new boolean[pixels.length];
    int len = 50;
    String[] fields = datacsv.split(",");
    for(int i = 0; i < fields.length; i++) {
       String[] cmp = fields[i].split(" ");
```

•

```
        cGrid[gridSize * (Integer.parseInt(cmp[1]) + len) + (Integer.parseInt(cmp[2]) + len)] =
true;     }
    new Thread(this).start();
    }
```

6.5 Capture of Seed Value

First, we get points of initial patterns randomly. And then we pass the initial population that consists of random patterns into Evolution method. After several rounds of running, we keep the initial population that can generate relatively better results in most cases. Finally, the initial population is stored as seed value so that we can reproduce good results in the future.

7 DATA ANALYSIS

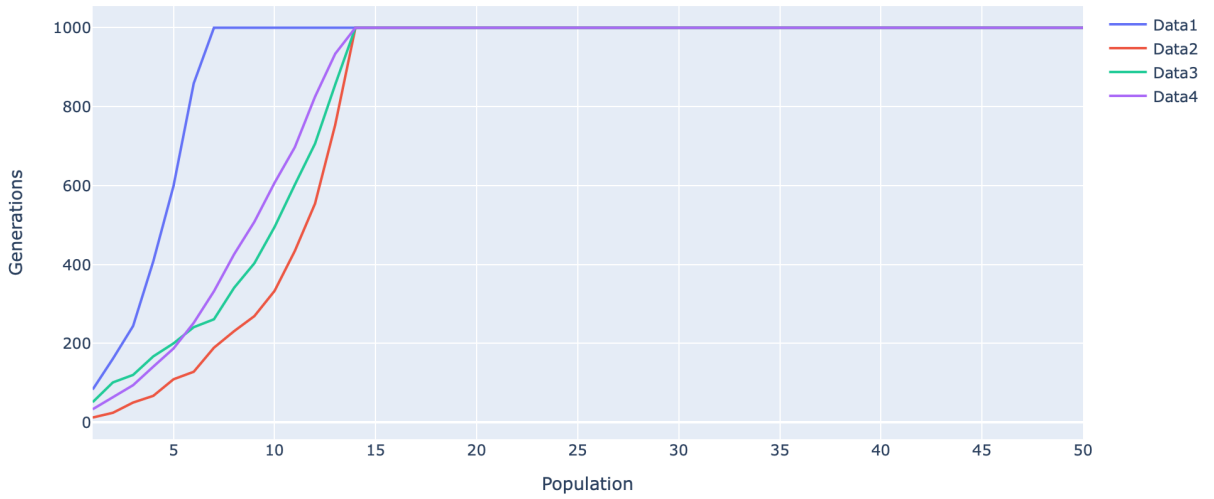

Figure 3. How population changes according to generation

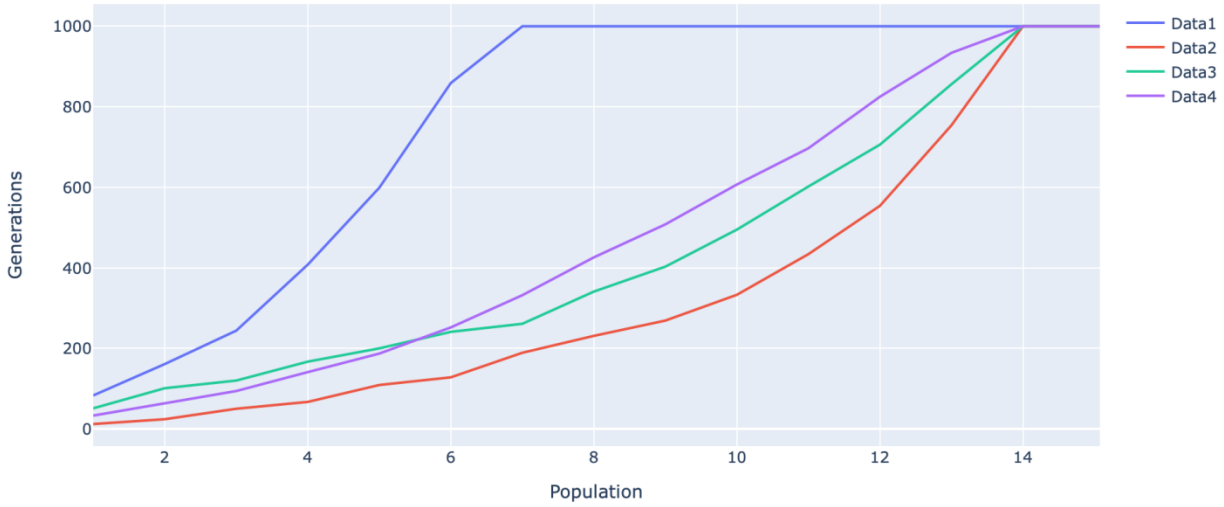

Figure 4. How population changes according to generation(part of Figure 3.)

We have tested our algorithm in 4 parameter configurations.
Data1:
public static int points = 100; // the number of points individually
public static int range = 20; // the range of x and y

•

public static int initialpopulation = 100; // the number of population
public static double rateofbreed = 0.5; // Proportion of organisms that survive and breed
public static double rateofmutation = 0.5; // Proportion of mutation

Data2:
public static int points = 20; // the number of points individually
public static int range = 10; // the range of x and y
public static int initialpopulation = 100; // the number of population
public static double rateofbreed = 0.5; // Proportion of organisms that survive and breed
public static double rateofmutation = 0.5; // Proportion of mutation

Data3:
public static int points = 10; // the number of points individually
public static int range = 5; // the range of x and y
public static int initialpopulation = 100; // the number of population
public static double rateofbreed = 0.5; // Proportion of organisms that survive and breed
public static double rateofmutation = 0.5; // Proportion of mutation

Data4:
public static int points = 10; // the number of points individually
public static int range = 5; // the range of x and y
public static int initialpopulation = 1000; // the number of population
public static double rateofbreed = 0.5; // Proportion of organisms that survive and breed
public static double rateofmutation = 0.5; // Proportion of mutation

We keep rate of breed and rate of mutation constant and unchanged. We change the number of points individually and range of x and y which create points. Also, we change the initial maxi-population.

But from the graph above, we find that figures of the data2, 3, 4 will meet the maxi-generation in population 14, but data 1 meets the maxi-generation in population 7. So, we assume that the number of initial points affects the result. Because more initial points put in Genetic Algorithms, there are more possibilities will happen. According to the mutation and selection methods, with more possibilities, they will be easier to find better initial population. That's why data1 meets more maxi-generation than the rest.

In the second, third, fourth experiment, we change the range of x and y. Though for the first 13 populations they act different, they all meet maxi-generation in population 14. And 36/50 of populations meet generation 1000.

•

To summarize, in data1 43/50 of our populations meet 1000 generations and in data 2 3 4 36/50 of our populations meet 1000 generations. So we assume that our designed genetic algorithm is effective.

8 CONCLUSION

In nature, Darwin leads us to the new world. It's ruled by the principle of evolution. Species are survival of the fittest. According to Conway's game of life, it simulates the environment competition and natural selection in a straight and easy way. In this two dimension world, we can accelerate the process of evolution. Switching the generationSpeed, from the birth of new population to the extinction of a species, we could get the result in a few minutes. Back to the problem we discussed in the report, how to find the best initial pattern and its offspring. So we tried to introduce genetic algorithms to solve this problem. Genetic Algorithms is simulation of Darwin's evolution theory.

First, we operate the program with random initial pattern(generation 0). And then we could get the first generation. We use mutation function (in our experiment, we set rateofmutaion is 0.5) to reproduce the next generation and also kill part of parent generations based on rateofbreed(in our experiment, we set rateofbreed is 0.5). Then we could get the new generation. What we do next is to continuingly keep good population and kill bad population in every round. There is a chance, though some good population will mutate to the bad one, they will be killed soon by natural selection, which is selection method in our project. So what we design is to keep the whole population evolves to a better one and will not be affected by the bad population.

All in all, we use the Genetic Algorithm to find fitter population and its initial pattern and offspring. Meanwhile, it meets the rules of the game of life and use the core idea of Genetic Algorithm. Also, we design our own evolution strategy. At last, if the program runs enough huge data and generations, we will get the result that is close to the optimal solution.

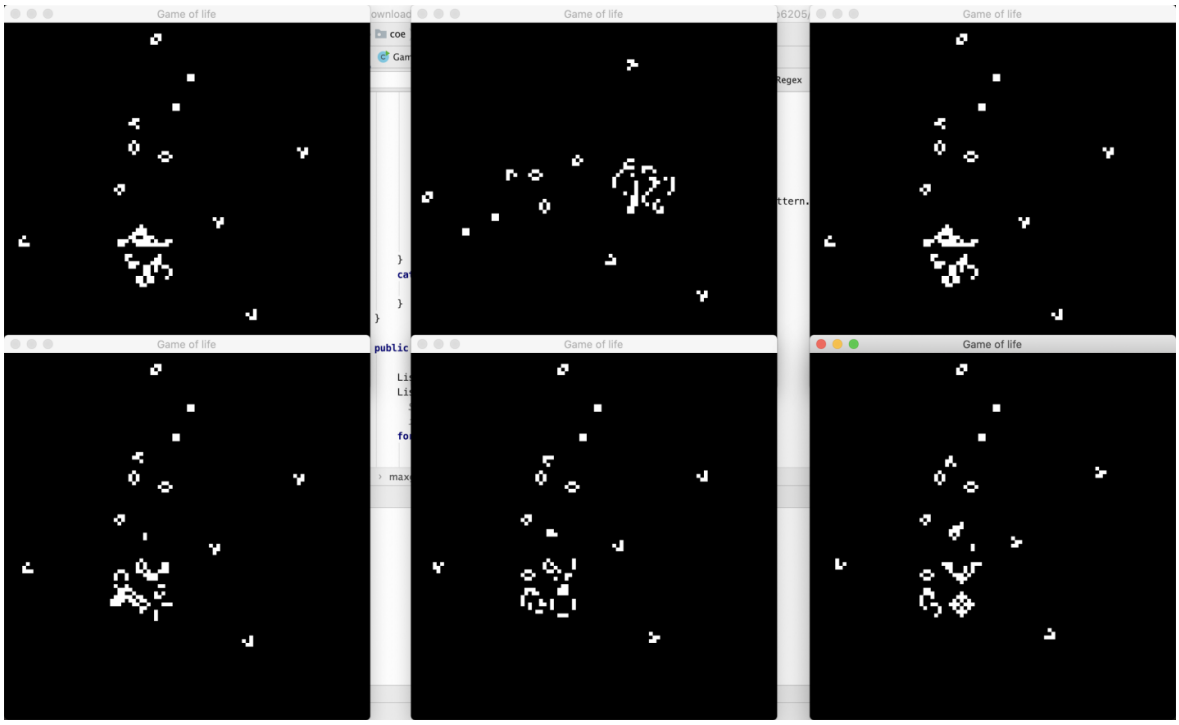

Figure 5. The ScreenShot of UI

The screenshot of UI after several thousand generations. We cannot show the changes beyond the grid even they are still growing.

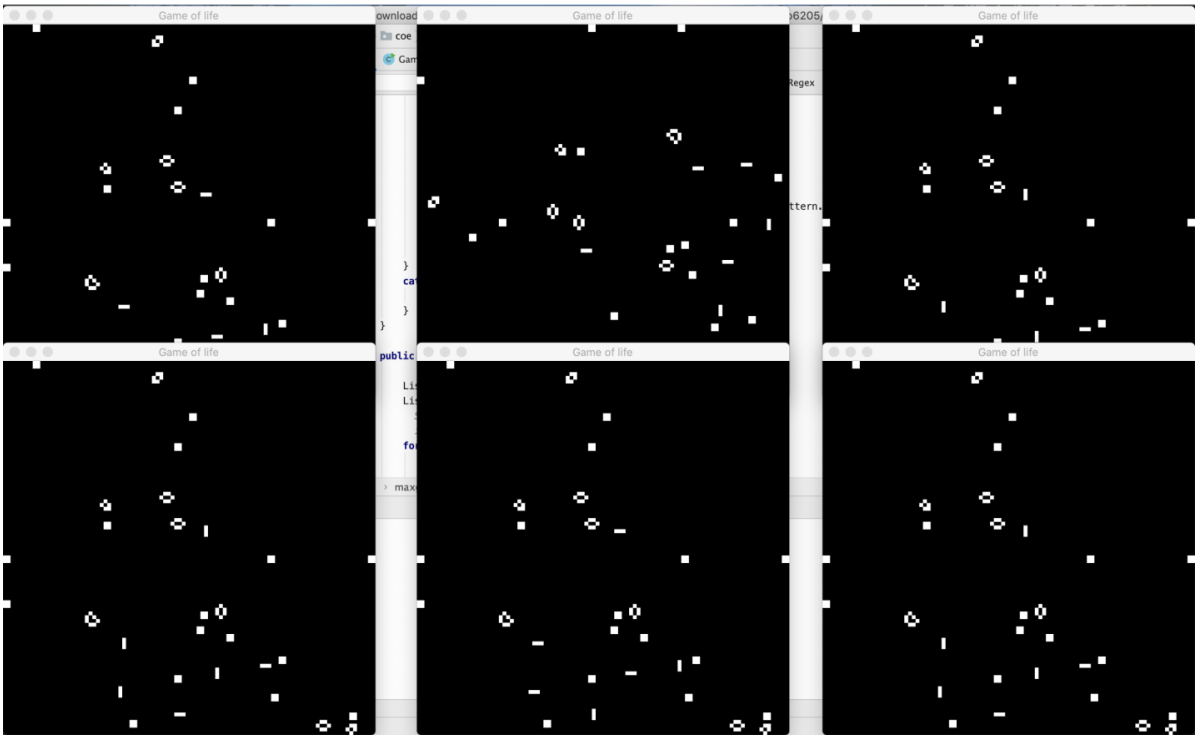

Figure 6.The ScreenShot of UI after several thousand generations

•

9 TESTING

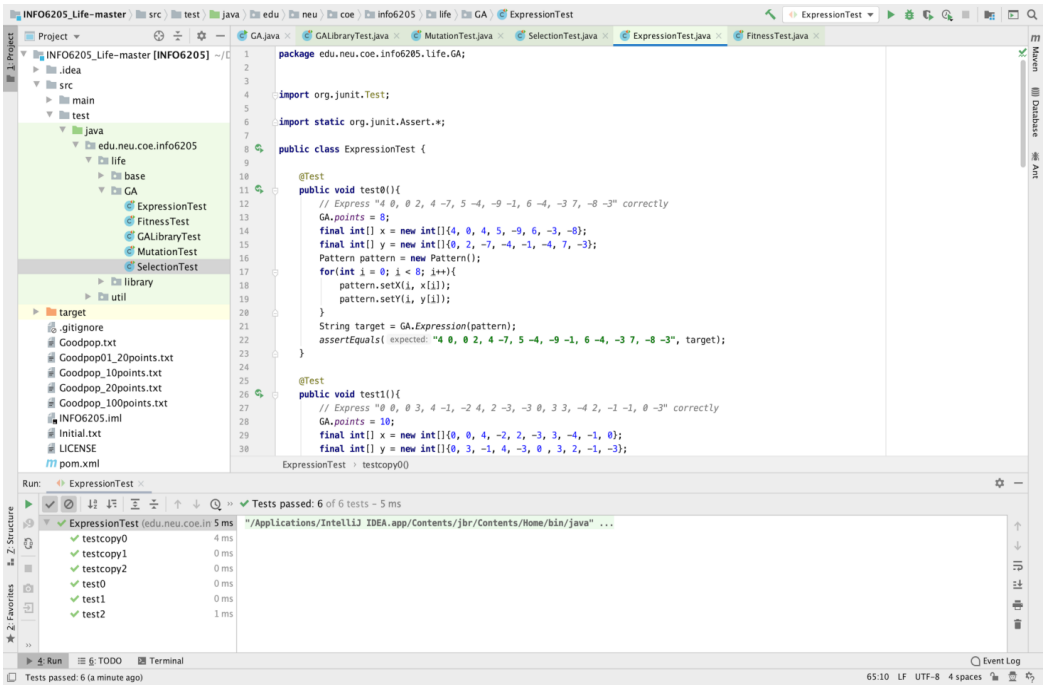

Figure 7. Test for Expression

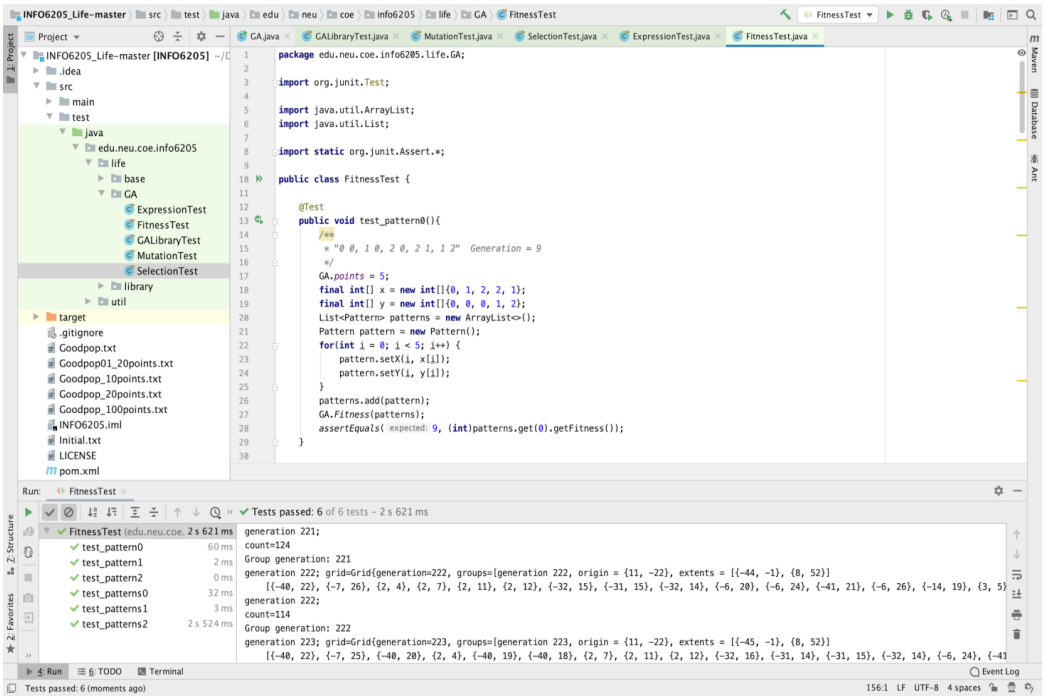

Figure 8. Test for Fitness

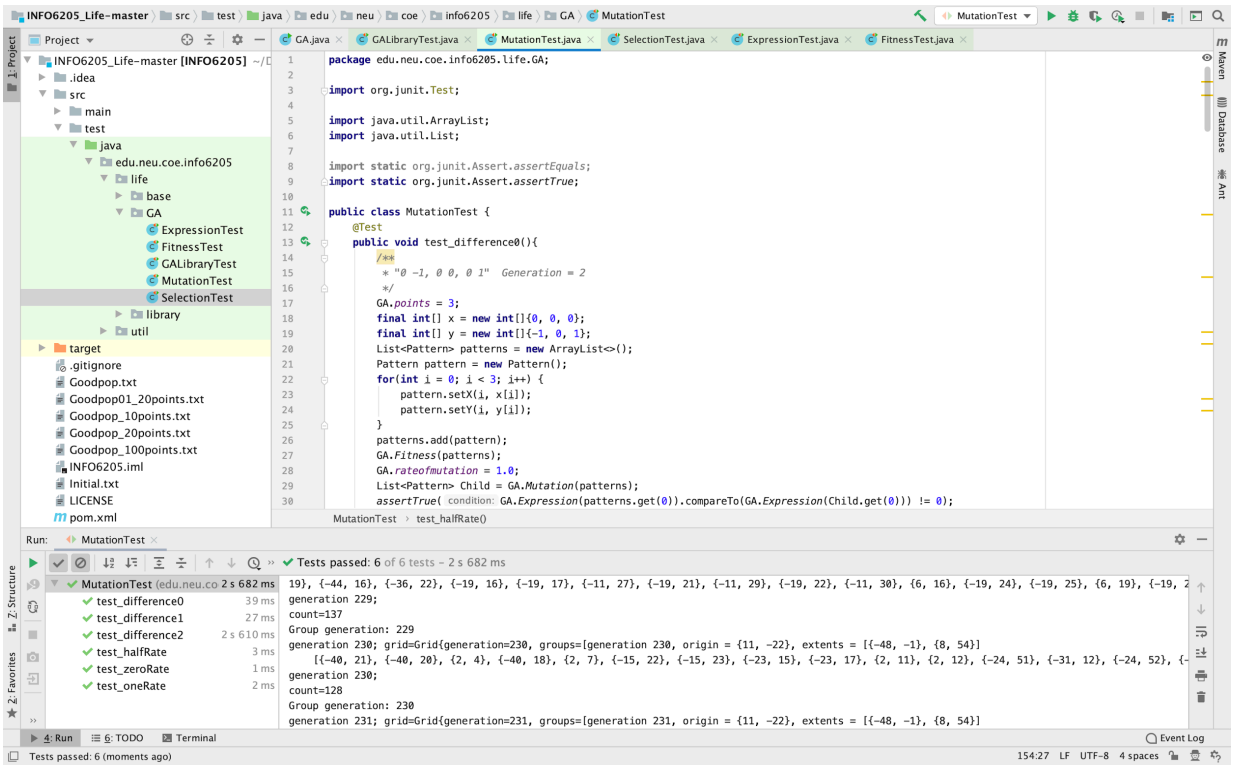

Figure 9. Test for Mutation

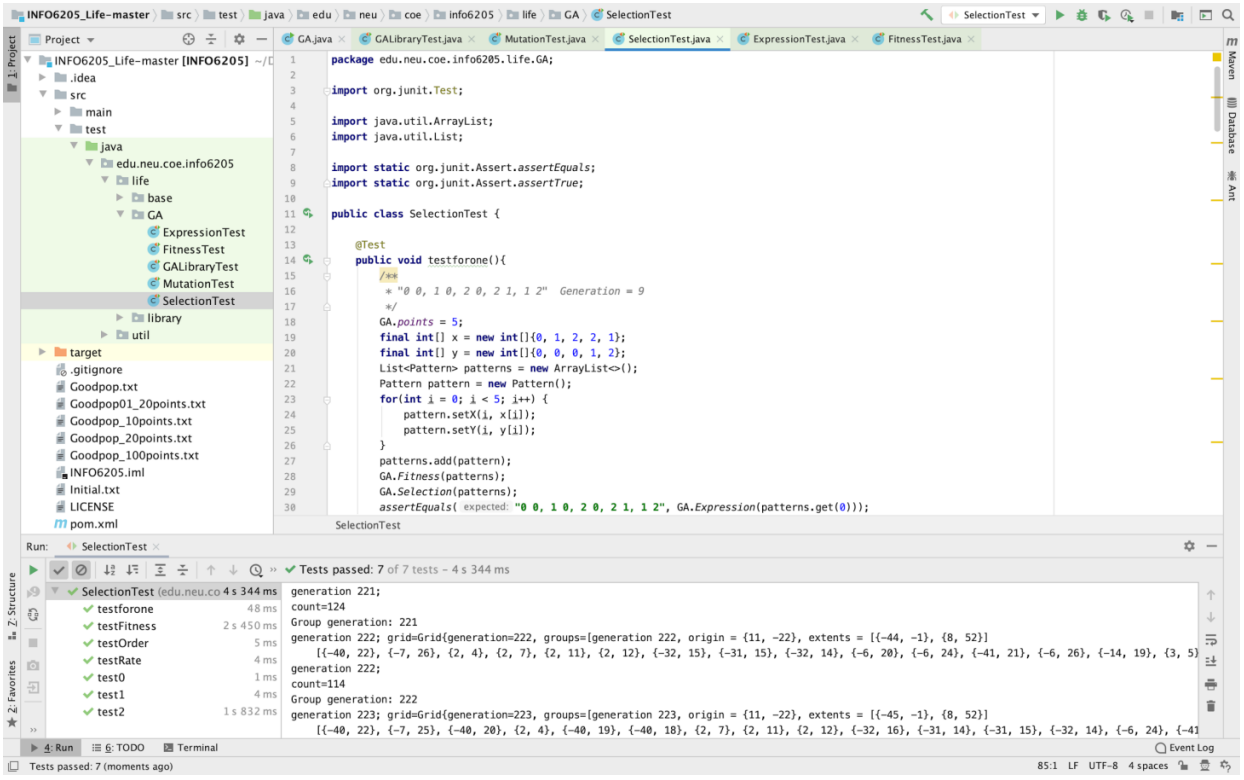

Figure 10. Test for Selection

•

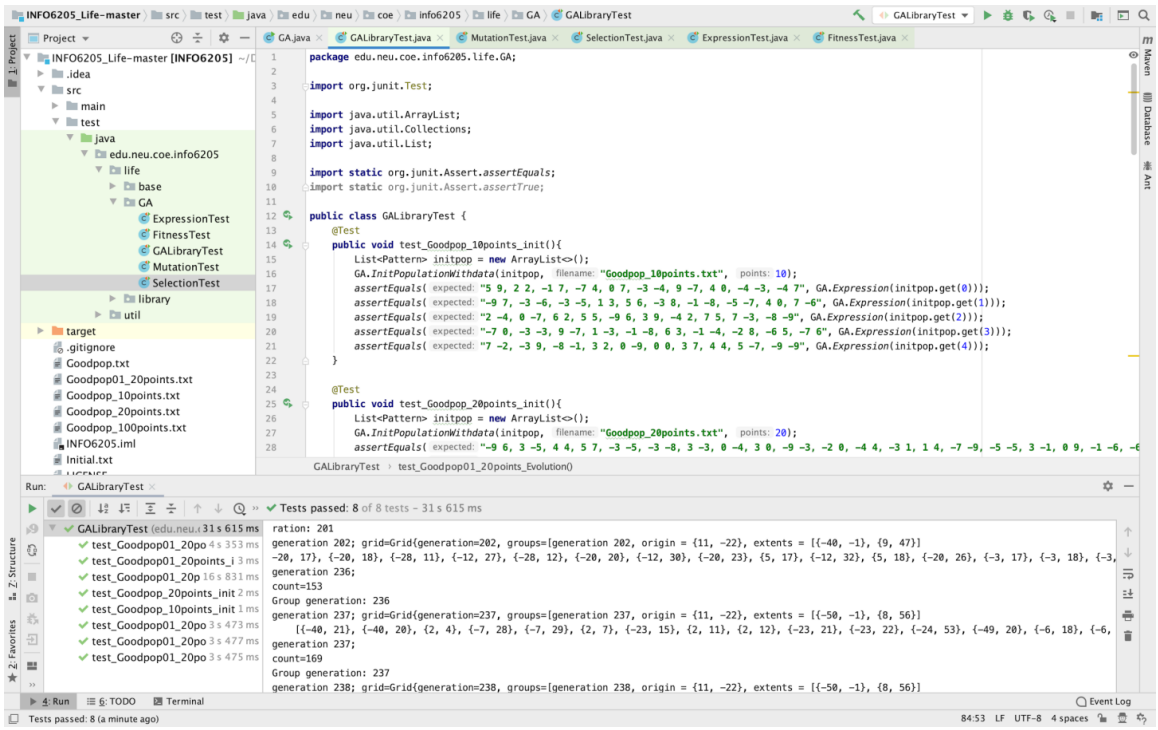

Figure 11. Test for GALibrary

We developed several classes to test the functions(Expression, Fitness, Mutation and Selection) and captured seed value(GALibrary). In each class, several specific groups of data were passed into corresponding methods. If the tests passed, the results meets our expectations, otherwise, we need to check the logic of code or improve our algorithm.

10 REFERENCE

[1]https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[2,3,4]https://s3.us-east-1.amazonaws.com/blackboard.learn.xythos.prod/5a3148150d016/18696896?response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27Life%2520Project%2520.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20191204T024646Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAIL7WQYDOOHAZJGWQ%2F20191204%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=5f9c96aaf442a0f1c7bcea4de79e63904757f2931658e0f5c7bdbe9185bc04e7