**Project Report**

**INFO 6205 Team 207**

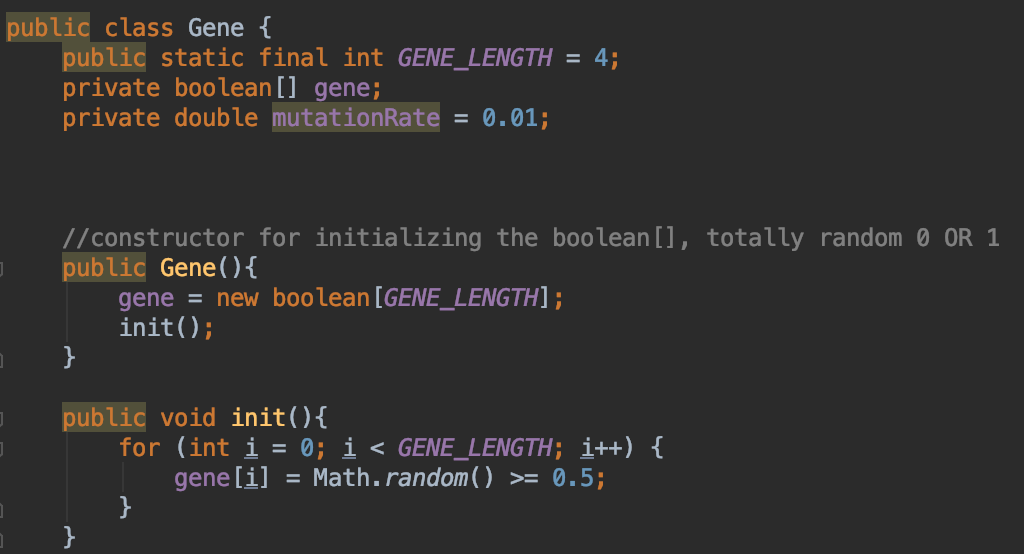
**Yezhi Miao 001401943 Ran Zhou 001448551**

**1.Aim of Project**

Using Genetic Algorithm to find a starting pattern which can last longer in Game of Life. The starting pattern is a string containing some coordinates.

**2.Basic Concepts of Genetic Algorithm**

**#Gene**

Gene is a basic element. In our project, we define a class named Gene, every instance of Gene can have a boolean[3 OR 4] 

In constructor, we use random figure to generate boolean[]. So for generation 0, all gene are totally randomly. The probabilities of generating 0 OR 1 are the same. The reason that we use boolean[] is that we all know that 0 represents false and 1 represents true.

Each gene is half of coordinates(x coordinates OR y coordinates).

**#Genotype**

Actually, gene is elements of genotype. So genotype is all the binary number of an individual, consisting of many Gene instances.

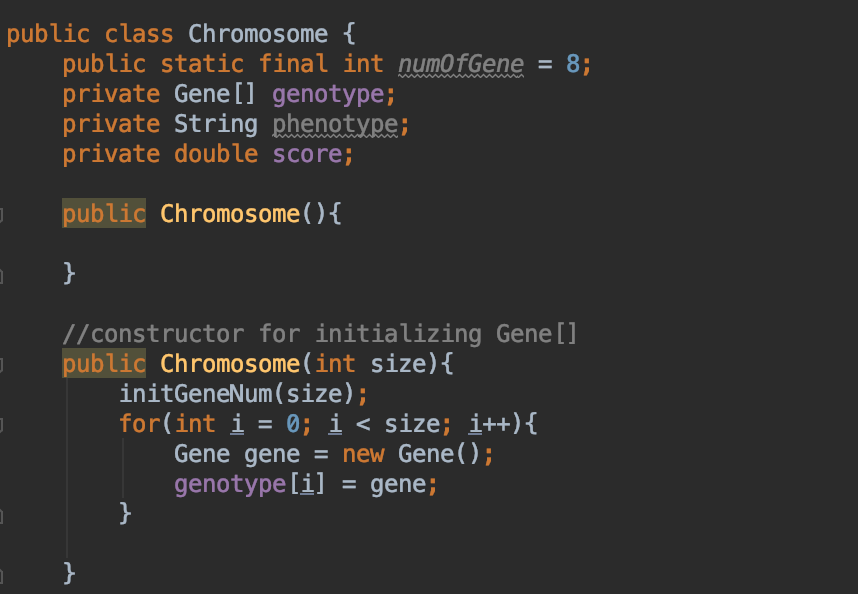
**#Phenotype**

In this case, phenotype is the starting pattern, which is translated from binary number to String.

Every Gene(half coordinate) can be turned from binary number to int value, as a result, an individual’s genotype can be converted into int value. We use these int values to form a String according to the required format in Class Library.

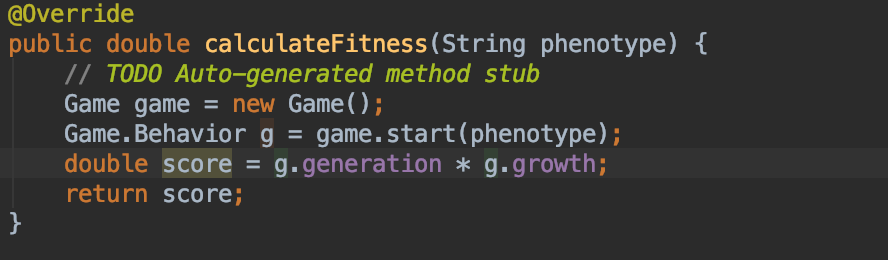
**#Chromosome**

We defined that an individual has a chromosome, each chromosome has 10 Gene[]. The number of Gene[] can be changed. Chromosome includes genotype and phenotype.



**#Fitness**

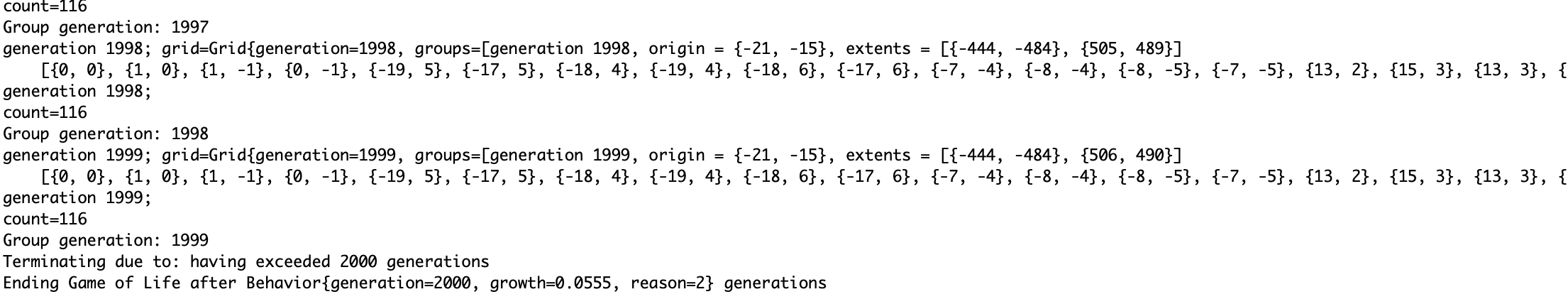
In our project, fitness is the how long the candidate can survive. In addition, we can also use the rate of growth to calculate the fitness:

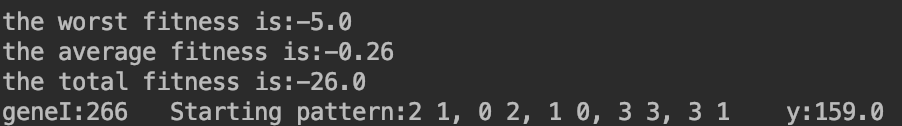


**#Problem**

We find our best solution, a string with 5 coordinates is : “2 1, 0 2, 1 0, 3 3, 3 1”

Here is the result:





The seed of this result(the generation0 of population) is:

3 3, 0 0, 1 0, 1 3, 3 2

0 -4, 3 3, -3 -4, -4 3, 2 0

-4 -4, 3 -3, 3 -4, -3 0, 1 3

-4 3, -3 -3, -4 3, 1 2, -4 1

2 3, -4 -4, 3 -4, 1 2, -4 3

3 -4, 0 -4, 2 -3, 0 3, 3 1

3 -4, 3 -4, 1 1, 1 -3, -4 3

-2 1, 1 1, 3 -4, 2 -2, 2 3

3 -2, -2 0, 3 -2, -4 1, 0 -3

3 -4, 3 0, 3 -4, 3 1, -3 3

2 3, -4 -4, 1 -3, 3 0, 1 2

-2 -3, 1 3, 2 3, 3 1, 2 1

3 -4, 0 -4, 1 0, -3 -4, 3 -3

0 3, 0 1, 3 0, 0 2, -2 0

3 1, -4 2, -4 3, 3 3, 2 1

-4 2, 3 3, 3 0, 3 3, 1 0

3 1, 0 -4, 1 2, 1 -2, 1 -2

-4 3, 3 3, 3 0, 1 -4, -3 -3

-4 3, 3 3, 2 -3, -4 3, 2 2

2 0, 0 0, 2 3, 3 -3, 0 2

2 3, 3 3, -4 3, -4 3, 3 3

-4 2, 3 1, -2 3, 3 2, 1 -3

0 3, -2 0, -3 3, 0 1, 3 -3

1 -4, -3 3, 1 2, -3 3, -3 3

3 3, 0 3, 1 -4, -2 0, 0 -3

-3 0, 3 -3, 1 -3, 1 0, 0 3

0 -4, 3 2, -4 3, 0 -3, 2 2

2 3, 2 3, 1 1, 3 0, 0 1

1 -3, -4 -4, -4 -3, -4 3, 3 -4

-3 1, 3 -2, 2 0, -4 -3, -4 3

3 0, 3 -4, -4 -2, -4 -4, 3 1

3 -3, 2 3, -3 -3, 3 1, 3 3

1 -2, -3 -3, 3 -2, -3 -3, 1 -3

-2 -4, 3 0, -2 -3, 1 3, 1 -4

-4 0, 1 3, 1 3, -4 -3, -2 2

3 -4, 2 -4, 3 1, -4 3, 2 -3

-2 1, 0 1, -3 3, 1 3, -2 3

-2 3, 2 -2, 3 -3, 0 3, 2 -2

3 2, 3 -4, -2 2, 1 0, 2 1

-4 1, -3 3, 3 -4, -4 1, -2 -3

-2 3, 2 1, 3 2, -4 2, 3 3

2 0, -3 0, -4 3, -3 -3, -4 -3

-4 -4, 2 -3, 3 2, 3 -3, 3 3

0 3, 3 1, 1 1, 3 3, 3 -2

3 3, 3 3, -3 -3, -2 -4, 0 0

0 -3, -4 -3, -4 1, -2 3, 2 -3

3 3, 3 -3, 1 3, 3 3, 3 2

-2 0, 2 -4, -4 3, 0 3, 3 -2

-2 -2, 3 3, -2 -2, 1 0, 2 -3

1 3, -3 -4, 3 0, -2 3, -3 -2

-4 0, -2 3, 0 3, -3 -3, -4 2

3 3, -3 3, 3 1, 3 3, -3 -3

-2 0, 2 0, 0 0, 3 -4, 1 -4

2 -3, 2 1, -2 0, 0 0, 1 1

0 1, 2 0, 0 -4, 2 -2, 1 3

3 -4, 0 -2, 2 3, 1 -3, 0 2

-4 2, -3 0, 3 1, 1 -4, 3 2

3 3, -3 3, -4 -4, 3 -3, 3 3

-2 1, 3 3, 0 0, 1 2, -2 -3

1 3, -3 -2, -3 3, -3 0, -2 2

3 -3, 3 0, -4 3, -3 2, 0 2

-2 -2, -2 2, 1 3, 3 2, 3 2

1 3, 0 1, 0 1, 3 0, -2 3

-3 2, -2 0, 3 -4, -4 -4, -4 3

0 -2, 1 -3, -3 2, 3 3, 2 2

0 2, 1 -4, -2 1, 1 2, 2 -3

3 -3, 1 -4, 0 -3, -3 0, 3 2

0 3, 0 3, -4 3, 2 2, -4 2

3 0, -4 3, 3 3, 3 -4, 1 1

1 2, -4 -3, -3 -4, -4 -4, -3 -3

1 1, 1 3, 3 3, 3 3, -4 3

2 -4, -4 0, -4 -3, 2 3, -4 3

-4 2, 1 1, 3 -2, 0 3, 1 2

0 -4, 3 -4, -4 2, 3 -3, 1 3

-4 2, -2 -3, 3 1, -3 2, 3 -3

2 2, -2 3, 1 -2, 2 2, 0 -2

3 -3, -3 -3, 1 2, -3 -4, 1 -4

0 -4, -2 0, 2 -2, 0 -4, -3 -3

1 1, 3 3, 0 2, -2 -4, 0 0

-3 3, 1 1, 3 -2, 2 0, -2 0

-4 1, 3 3, 1 3, 3 -3, -3 3

2 -2, 3 3, -4 3, 2 3, 3 2

1 3, 3 -2, 3 3, -2 2, -3 1

2 -3, 2 -4, 1 -2, -2 3, 1 3

-3 3, 0 -4, -3 -3, 1 3, -4 -2

-3 0, 3 -3, -4 3, 3 1, -3 1

-2 3, 3 2, 0 1, 2 2, 1 -4

2 -3, 0 3, 3 0, 2 3, 3 3

1 2, -3 -4, -3 3, 3 -3, 1 3

-4 2, 3 -4, 3 3, -3 3, 1 3

3 -4, 2 2, 3 3, -2 1, 3 -3

3 3, -3 2, 0 0, 2 -4, -2 0

0 2, -4 -4, 3 -4, 2 2, 1 3

1 3, -4 -3, 2 1, 0 -3, 0 0

3 2, -3 0, -4 3, 3 -4, 2 -4

-3 2, -4 1, 0 2, -3 2, 2 1

-3 -2, -3 1, 2 -2, 0 -2, -2 0

2 3, -4 -2, -2 -3, 2 3, 0 -4

-4 3, 1 0, 1 3, 1 -3, 0 3

1 3, -4 -3, 1 2, 3 0, -2 3

**3.Framework of Detailed Implementation**

Class Gene:

public class Gene {  
 public static final int *GENE\_LENGTH* = 3;  
 private boolean[] gene;  
 private double mutationRate = 0.01;  
  
 //constructor for initializing the boolean[], totally random 0 OR 1  
 public Gene(){  
 gene = new boolean[*GENE\_LENGTH*];  
 init();  
 }  
  
 public void init(){  
 for (int i = 0; i < *GENE\_LENGTH*; i++) {  
 gene[i] = Math.*random*() >= 0.5;  
 }  
 }  
  
 //get int value of boolean[] gene,convert binary to int  
 public int getNum() {}  
  
 //if random figure < mutation Rate, 0 to 1 OR 1 to 0$  
 public void mutation() {}  
  
 public boolean[] getGene(){}  
  
 public void setGene(boolean[] b){}  
}

Class Chromosme:

public class Chromosome {  
 public static final int *numOfGene* = 8;  
 private Gene[] genotype;  
 private String phenotype;  
 private double score;  
  
 public Chromosome(){}  
  
 //constructor for initializing Gene[]  
 public Chromosome(int size){}  
  
 public void initGeneNum(int size){}  
  
 //clone gene for generate next generation  
 public static Chromosome clone(final Chromosome c) {}

//let two Chromosome exchange part of gene  
 public static List<Chromosome> genetic(Chromosome p1, Chromosome p2) {}

//every gene bit has probability to mutate  
 public void mutation(){}

//get phenotype  
 public String getPhenotype(){}

//score is the fitness of candidate  
 public double getScore() {}  
  
 public void setScore(double score) {}

//check it phenotype has duplicates, duplicates can cause some error in Class Game, some duplicates can last forever  
 public boolean ifDuplicate(){}  
  
 public Gene[] getGenotype() {}  
}

Class GeneticAlgorithm:

public abstract class GeneticAlgorithm {  
  
 private List<Chromosome> population = new ArrayList<>();//种群  
 private List<Chromosome> farPopulation = new ArrayList<>();  
 private int popSize = 100;//size of pupulation  
 private int geneSize;//max length of gene  
 private int maxIterNum = 500;//max number of evolution  
 private double mutationRate = 0.01;//probability of mutation  
 private int maxMutationNum = 3;//max length of mutation  
 private int generation = 0;//current generation of population  
  
 private double bestScore;//best fitness  
 private double worstScore;//worst fitness  
 private double totalScore;//total fitness  
 private double averageScore;//average fitness  
  
 private String x; //best solution(phenotype) in all population  
 private double y; //best fitness   
 private int geneI;//the generation of population which contain x & y  
 private List<Chromosome> generation0 = new ArrayList<>();//save seed  
  
  
 public GeneticAlgorithm(int geneSize) {}  
  
 public void init() {}

//calculate each candidate’s fitness and get best,worst average score  
 private void calculateScore() {}  
  
 private void setChromosomeScore(Chromosome c) {}  
  
 //get phenotype of a chromosome

public abstract String changeX(Chromosome c);

//using abstract method to calculate candidate  
 public abstract double calculateFitness(String phenotype);  
  
 //select Chromosome with hight fithness

public Chromosome selectChromosome(){}  
  
 //generate a population with the same size

private void generateagain(){}  
  
 //the generation evolve by selecting and mutation

private void evolve() {}  
  
 private void mutation() {}  
  
 public void caculte() {}  
  
 //print information of best worst ave score and best solution

private void print() {}  
  
 public void setPopulation(List<Chromosome> population) {}  
  
 public void setPopSize(int popSize) {}  
  
 public void setGeneSize(int geneSize) {}  
  
 public void setMaxIterNum(int maxIterNum) {}  
  
 public void setMutationRate(double mutationRate) {}  
  
 public void setMaxMutationNum(int maxMutationNum) {}  
  
 public double getBestScore() {}  
  
 public double getWorstScore() {}  
  
 public double getTotalScore() {}  
  
 public double getAverageScore() {}  
  
 public String getX() {}  
  
 public double getY() {}  
  
 public void setBestScore(double bestScore) {}  
  
 public void setWorstScore(double worstScore) {}  
  
 public void setTotalScore(double totalScore) {}  
  
 public void setAverageScore(double averageScore) {}  
  
 public List<Chromosome> getPopulation() {}  
}

Class Driver:

public class Driver extends GeneticAlgorithm {  
  
 public static final int *NUM* = 1 << 10;  
  
 public Driver() {  
 super(10);  
 }  
  
 @Override  
 public String changeX(Chromosome chromosome){}  
  
 @Override  
 public double calculateFitness(String phenotype) {  
 // *TODO Auto-generated method stub* Game game = new Game();  
 Game.Behavior g = game.start(phenotype);  
 double score = g.generation \* g.growth;  
 return score;  
 }  
  
 public static void main(String[] args) {  
 Driver test = new Driver();  
 test.caculte();  
  
  
  
 }  
}

This class has main method and we should run this class.