INFO6205 Program Structure & Algorithms —Genetic Algorithms Project

Solve a 2D Maze Problem with Genetic Algorithms

Group Members:

Yi Liu

Hao Zuo

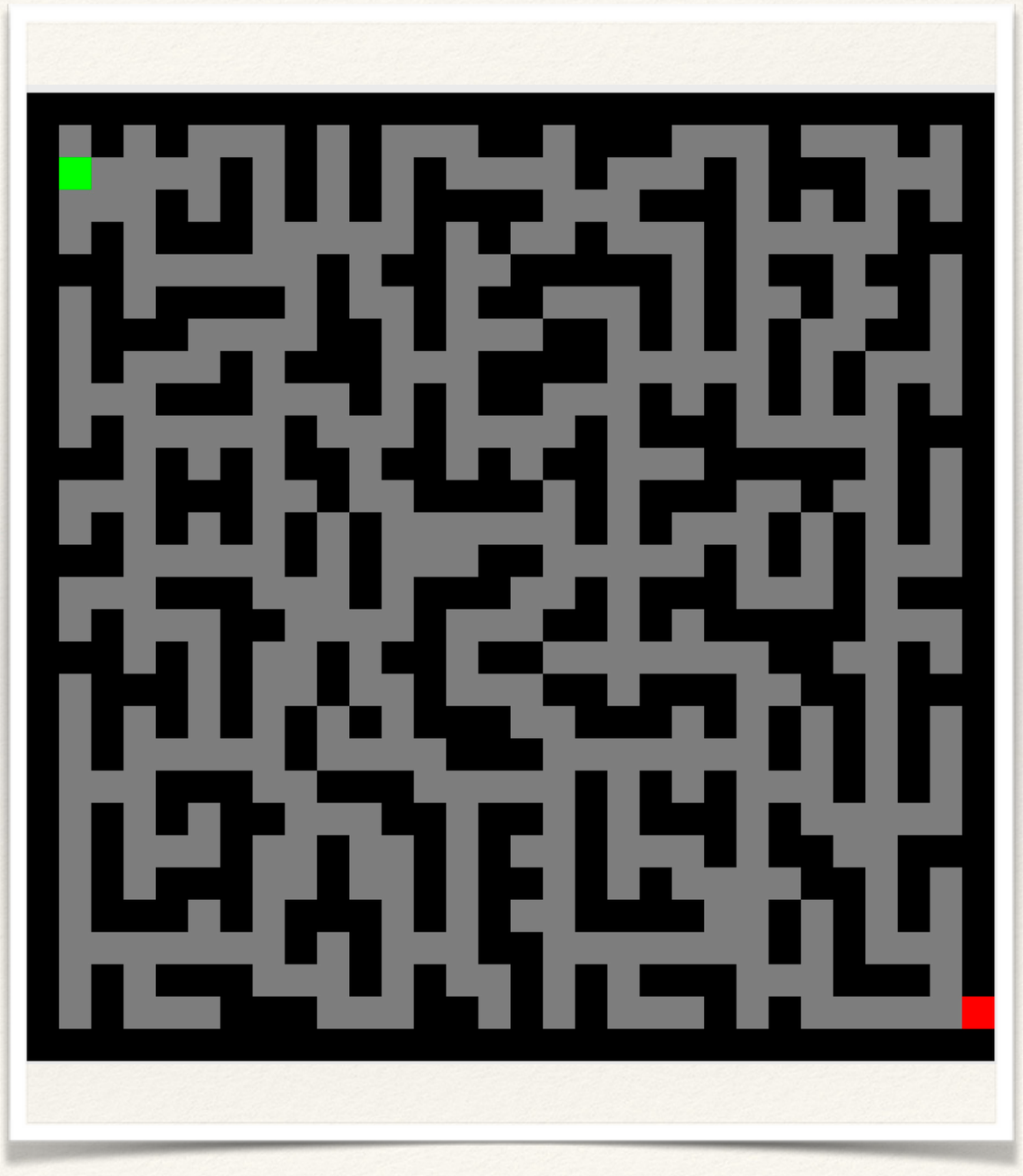
Yuxuan Xiong

Catalogue

- Problem Description
- Code Description
- Analysis & Findings
- Results
- Conclusions

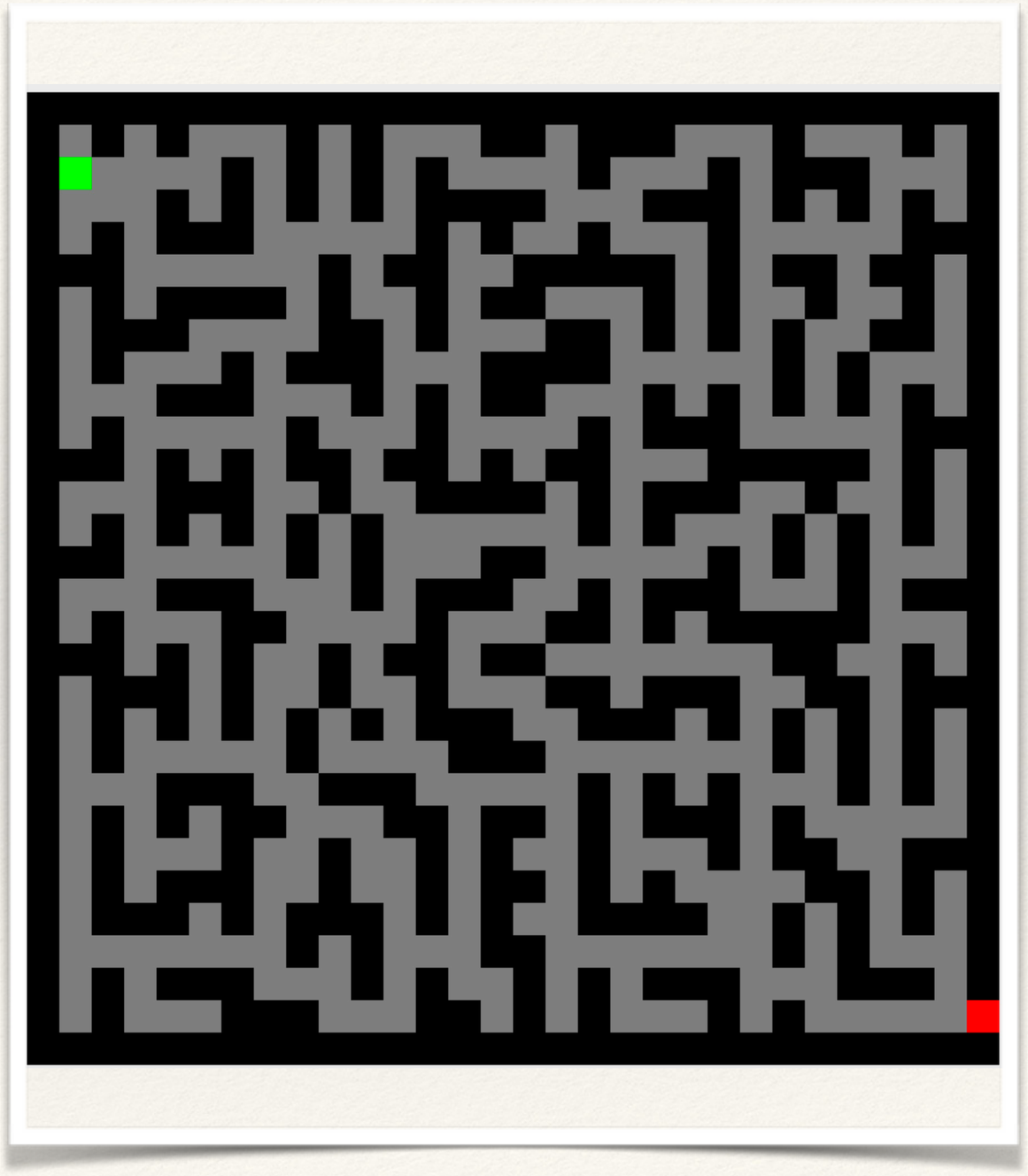
Problem Description

- Our group's goal is to find a way from start point(the green block) to end point(the red block) with Genetic Algorithms in the maze(as shown in picture).
- The maze is stable(size: 30*30) which is inputed by tester.
- The black block means "wall" which can't be passed through, and the gray block means the available path .



Problem Description

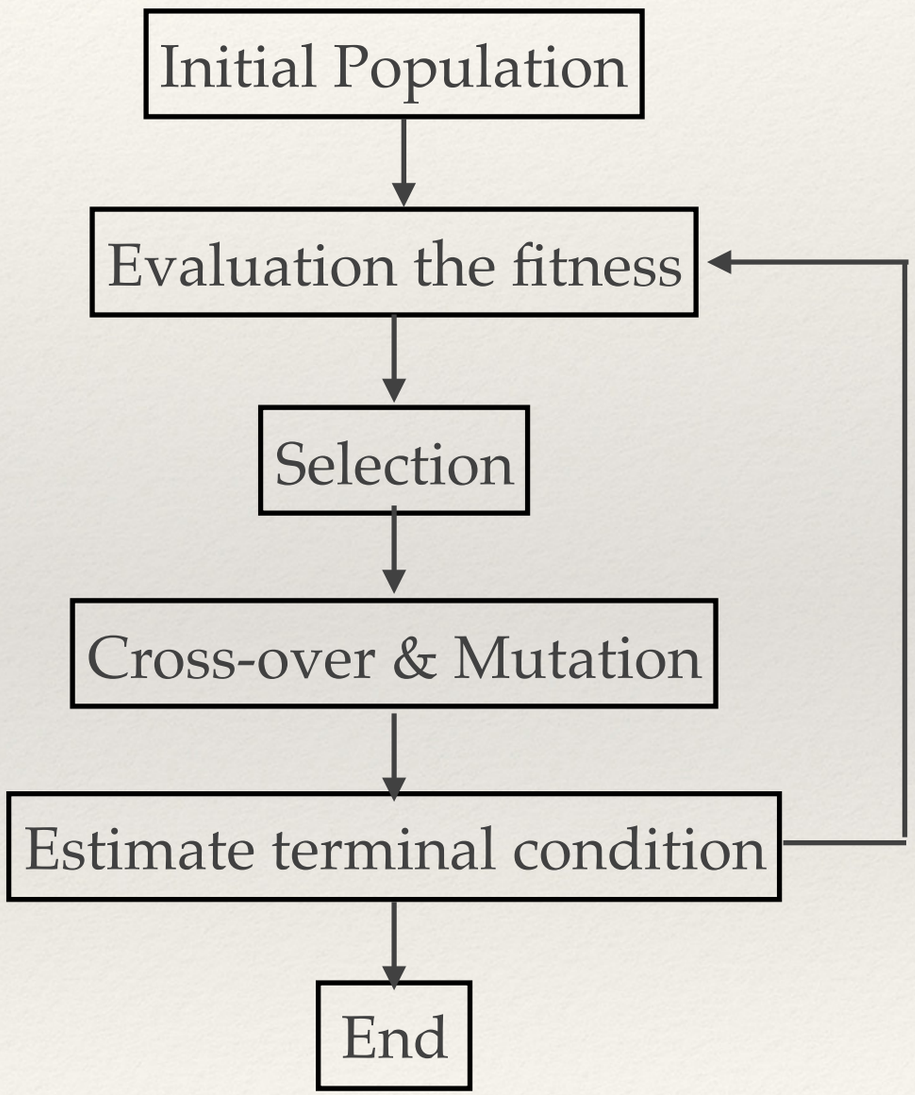
- We aim to find a way between start and end point in the maze with the Genetic Algorithms, not the shortest path.
- Sometimes, the program can't get the path between entrance and exit(because of the limitation of maximum number of generations). But the program should output the best route among all individuals.



The Procedure in GA

Configuration Parameters

- Initial population : 1000
- Proportion of organisms that survive:0.5
- Fecundity of mating : 2(2 offspring per pair)
- Generations to reproductive maturity: 1
- Max number of generations: 10000



The Expression of Genotype

- In this project, each individual has a chromosome which is a int[] type of numbers with 0 and 1.
- Each two numbers in the chromosome means one step.
- For example, if an individual has 1000 numbers in its chromosome, it means it can move 500 steps in the maze.

```
public class Individual {
    public int length;
    public int[] chromosome;
    public double score;
    public int getlength() {
        return length;
    }
    public void setlength(int length) {
        this.length=length;
    }
    public int[] getgene() {
        return chromosome;
    }
    public void setgene(int gene[]) {
        this.chromosome=gene;
    }
    public double getscore() {
        return score;
    }
    public void setscore(double score) {
        this.score=score;
    }
    public Individual(int length){
          this.length = length;
          chromosome = new int[length];
          Random random = new Random();
          for(int i=0;i<length;i++){
           chromosome[i] = random.nextInt(2);
          }
          this.score=0;
    }
}
```

The Expression of Genotype

Consider the gene of a individual is “00011011”.

It means this individual must move “up, down, left, right” from the start point.

However, if there is a “wall”(black block in the map) in the next step, the individual won’t move in this step. In another word, This single gene is not expressed.

```
public void route(int[] chromosome) {
    int x= maze.Xstart;
    int y= maze.Ystart;
    for(int i=0;i<chromosome.length;i+=2) {
        if(chromosome[i]==0 && chromosome[i+1]==0) {
            if(x>0&&maze.map[x-1][y]==0) {
                x--;
            }
        }//move down
        else if(chromosome[i]==0 && chromosome[i+1]==1) {
            if(x<maze.map_height && maze.map[x+1][y]==0) {
                x++;
            }
        }//move up
        else if(chromosome[i]==1&& chromosome[i+1]==0) {
            if(y>0 && maze.map[x][y-1]==0) {
                y--;
            }
        }//move left
        else {
            if(y<maze.map_width && maze.map[x][y+1]==0) {
                y++;
            }
        }//move right
    }
}
```

Initialization

- We generated 1000 individuals which with different chromosomes. Then, we used “fittest” to record the best one in this generation. At last, we put all of these 1000 individuals in a group which called “individualGroup”.

```
public void init() {
    maze.readDataFile();
    for(int i=0;i<individual_size;i++)
    {
        Individual chromosome =new Individual(chromosomeLength);
        double distance=distance(chromosome.getgene());
        if(distance > fittest.getscore()){
            fittest = chromosome;
          }
        chromosome.setscore(distance);
        individualGroup.add(chromosome);
    }
}
```

The Evaluation of the Fitness

In this project, we used the distance to the end point to represent the fitness. Each individual has a score(you can see the details how to get it in the picture).

The biggest score in this project is 0.5, which means if one individual's score equaled to 0.5, this individual would have arrived the exit in the maze.

```
public double distance(int[] chromosome) {
    double score=0;
    int x= maze.Xstart;
    int y= maze.Ystart;
    for(int i=0;i<chromosome.length;i+=2) {
        if(chromosome[i]==0 && chromosome[i+1]==0) {
            if(x>0&&maze.map[x-1][y]==0) {
                x--;
            }
        }//move up
        else if(chromosome[i]==0 && chromosome[i+1]==1) {
            if(x<maze.map_height && maze.map[x+1][y]==0) {
                x++;
            }
        }//move down
        else if(chromosome[i]==1&& chromosome[i+1]==0) {
            if(y>0 && maze.map[x][y-1]==0) {
                y--;
            }
        }//move right
        else {
            if(y<maze.map_width && maze.map[x][y+1]==0) {
                y++;
            }
        }
        //move left
    }
    double absx=Math.abs(x-maze.Xend);
    double absy=Math.abs(y-maze.Yend);
    score =1/(absx+absy+1);
    return score;
}
```

The Evaluation of the Fitness

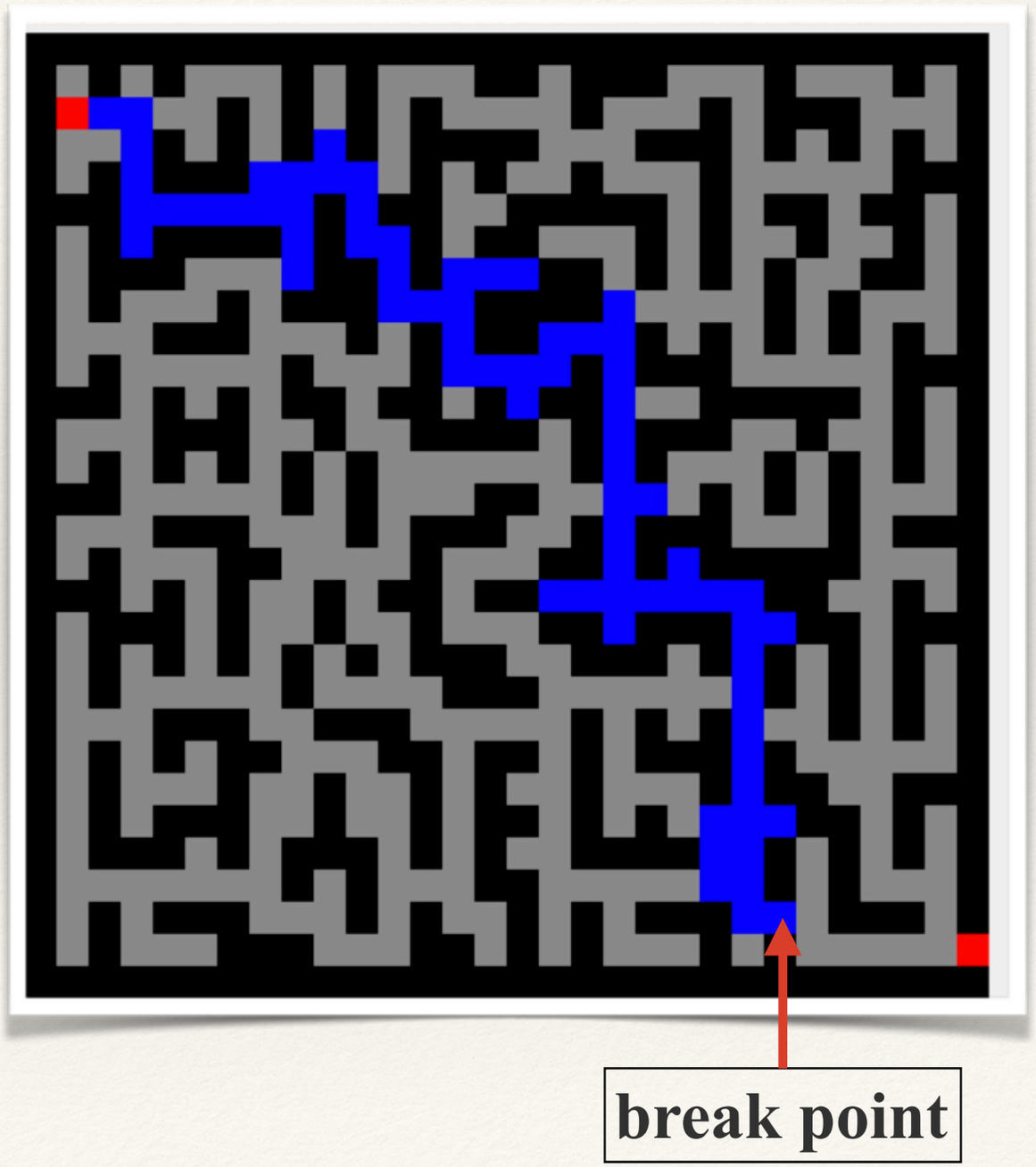
- Example:

The coordinate of the exit is(28,29) and the final coordinate of one individual is (27,23) which is shown in picture.

The distance in X direction is absX=1; and the distance in Y direction is absY=6.

The score is 1/1+6+1=0.125

The higher of the score represents the closer to the distention.



Selection

In this project, we used roulette wheel selection method to select individuals with high fitness. Each individual has a probability to be selected. The individual with a high fitness score has a high probability to be chosen.

In the code, the selectOperation() function return a number of the key value of the “survival”. And the survival() function is a loop of selection to select 500 individuals in the individualGroup(contains all individuals).

```java
public int selectOperation(List<Individual> list) {
    double randomNum = 0;
    double sum=0;
    int result = 0;
    double[] adaptiveValue = new double[individual_size];
    for(int i=0;i<individual_size;i++){
        Individual gene =list.get(i);
        double x=gene.getscore();
        adaptiveValue[i]=(x*x);
        sum+=adaptiveValue[i];
    }
    for (int i = 0; i < individual_size; i++) {
           adaptiveValue[i] = adaptiveValue[i] / sum;
    }

    for (int i = 0; i < individual_size; i++) {
           randomNum = Math.random();
    }
    sum=0;
    for (int j = 0; j < individual_size; j++) {
        if (randomNum > sum&& randomNum <= sum + adaptiveValue[j]) {
            for(int i=0;i<individual_size/2;i++) {
                result=j;
            }
         break;
        }
        else {
         sum += adaptiveValue[j];
        }
    }
    return result;
}
public int[] survival() {
    int[] survival =new int[individual_size/2];
    for(int i=0;i<survival.length;i++) {
        survival[i]=-1;
    }
    int count=0;
    while(count<individual_size/2) {
        Evolver main = new Evolver();
        int x=main.selectOperation(individualGroup);
        for(int i=0;i<survival.length;i++) {
            if(x==survival[i]) {
                break;
            }
            else if(x!=survival[i]&&i==survival.length-1) {
                survival[count]=x;
                count++;
            }
        }
    }
    return survival;
}
```

Crossover and Mutation

Firstly, We chose a pair of survival individuals as “mother” and “father”. Then, each parents will generate two “babies” called c1 and c2 in the code. In this part, we defined two rates: “mate rate” and “mutation rate”. It means the children’s chromosomes have probability to crossover and mutate(the best rate we will discuss later). Finally, we put “father”, “mother”, two “children” in a new group.

```
public List<Individual> evolution(int mother,int father) {
      List<Individual> partgenegroup = new ArrayList<>();
      Individual p1 =  individualGroup.get(mother);
      Individual p2 = individualGroup.get(father);
      Individual c1 = new Individual(chromosomeLength);
      Individual c2 = new Individual(chromosomeLength);
      int[] chromosome1 = new int[chromosomeLength];
      int[] chromosome2 = new int[chromosomeLength];
      for(int i=0;i<chromosomeLength;i++){
           chromosome1[i] = p1.getgene()[i];
           chromosome2[i] = p2.getgene()[i];
      }
      Random random=new Random();
      double r = random.nextDouble();
      if(r <= mate_rate){
       int n = random.nextInt(chromosomeLength);
       for(int i=n;i<chromosomeLength;i++){
        int tmp = chromosome1[i];
        chromosome1[i] = chromosome2[i];
        chromosome2[i] = tmp;
       }
      }
      r = random.nextDouble();
      if(r <= mutation_rate){
           int n = random.nextInt(chromosomeLength);
           if(chromosome1[n] == 0){
            chromosome1[n] = 1;
           }
           else{
            chromosome1[n] = 0;
           }
           if(chromosome2[n] == 0){
            chromosome2[n] = 1;
           }
           else{
            chromosome2[n] = 0;
           }
          }
      c1.setgene(chromosome1);
      c2.setgene(chromosome2);
      double score1 = distance(c1.getgene());
      double score2 = distance(c2.getgene());
      if(score1 >fittest.getscore()){
          fittest = c1;
      }
      if(score2 >fittest.getscore()){
          fittest = c2;
      }
      c1.setscore(score1);
      c2.setscore(score2);
      partgenegroup.add(c1);
      partgenegroup.add(c2);
      partgenegroup.add(p1);
      partgenegroup.add(p2);

      return partgenegroup;
}
```

The Termination Condition

There are two termination conditions.

1. The number of the generations reach the maximum.

2. Find route to the exit.

Whatever the program find the route or not. There will be one individual with the highest fitness score. Our group used a Java Panel Interface to show this route in the maze.

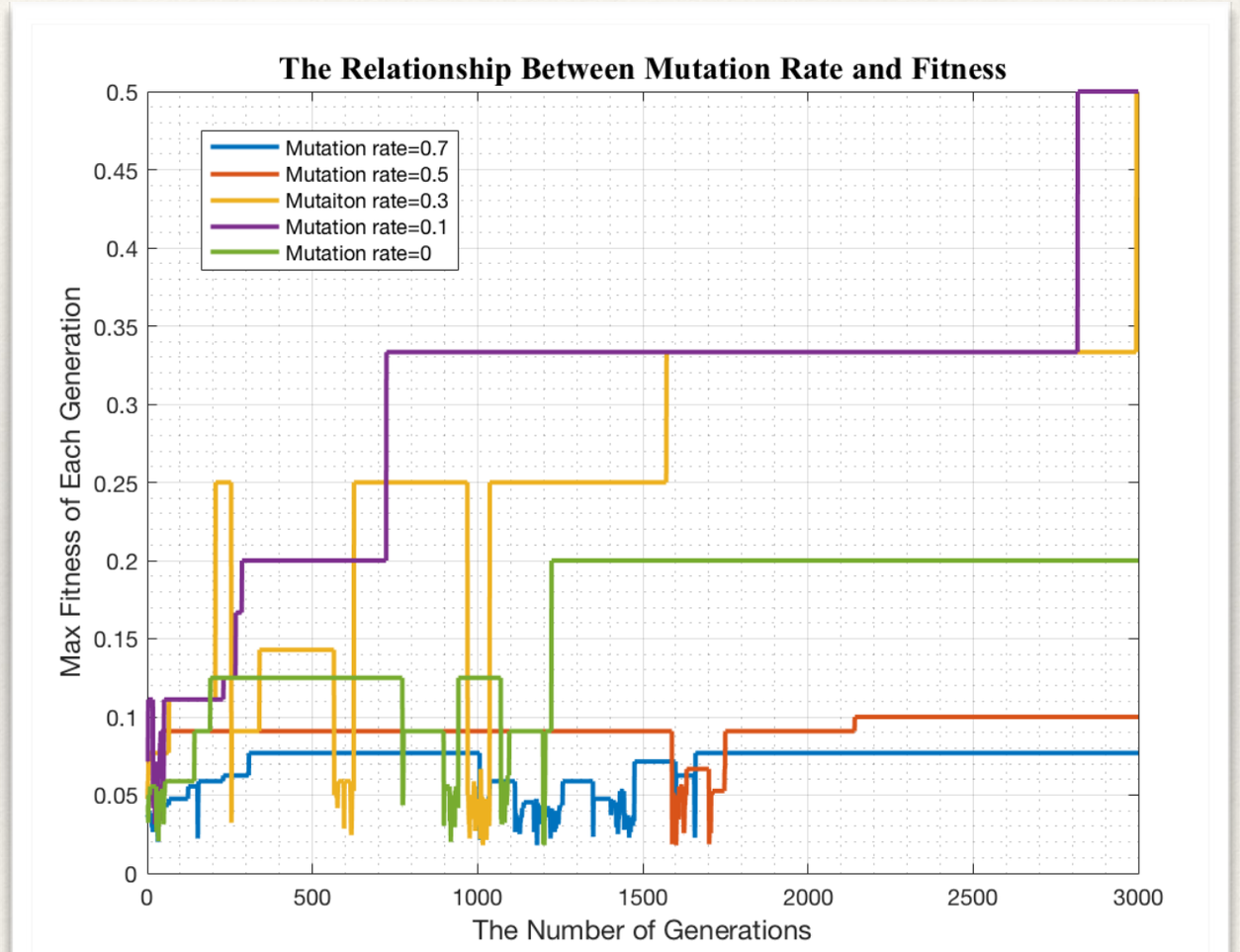
Analysis & Findings

In the Genetic Algorithms some factors like Mutation rate, Crossover rate and Gene length will affect the final results. Our group designed some tests to find the best value of these parameters.

The Relationship Between Mutation Rate and Fitness Score

The picture shows the relationship between different mutation rates and fitness scores within 3000 generations.

When the mutation rate is high, the fitness score fluctuated obviously with the number of generations increased. However, when the mutation rate is low, like the green line in the picture, the fitness score remain the same after 1200 generations.



Crossover rate=0.9, Gene Length=1000

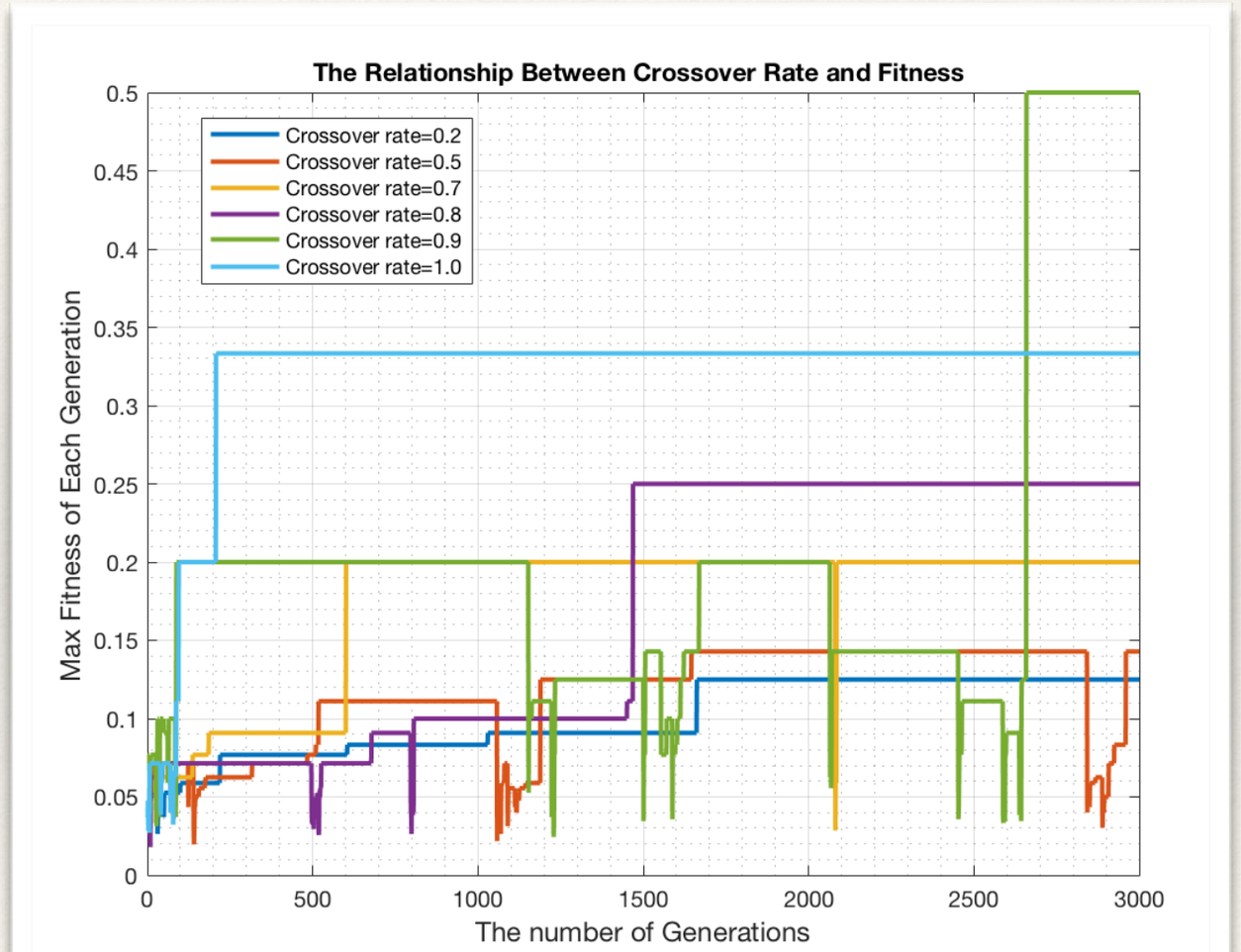
The Relationship Between Mutation rate and Fitness Score

- When the mutation rate is too high, it will make the individual with the best fitness mutate to a bad one easily. Finally, it leads no individual gets to the exit within 3000 generations.
- Moreover, when the mutation rate is too low, it makes the route fall into a dead end and can’t get out in this maze problem. In another word, it will make the calculation trapped in local optimal solution.
- In conclusion, from the picture we can get that the suitable mutation rate is between 0.1-0.3. Finally, we select 0.2 as the mutation rate.

The Relationship Between Crossover rate and Fitness Score

The picture shows the relationship between different crossover rates and fitness scores within 3000 generations.

When the crossover rate is low, the fitness score fluctuated unobviously with the number of generations increased. However, when the crossover rate is high, like the green and blue line in the picture, the fitness score is reached into 0.5, which means the individual reached the exit of maze.



Mutation rate=0.2, Gene Length=1000

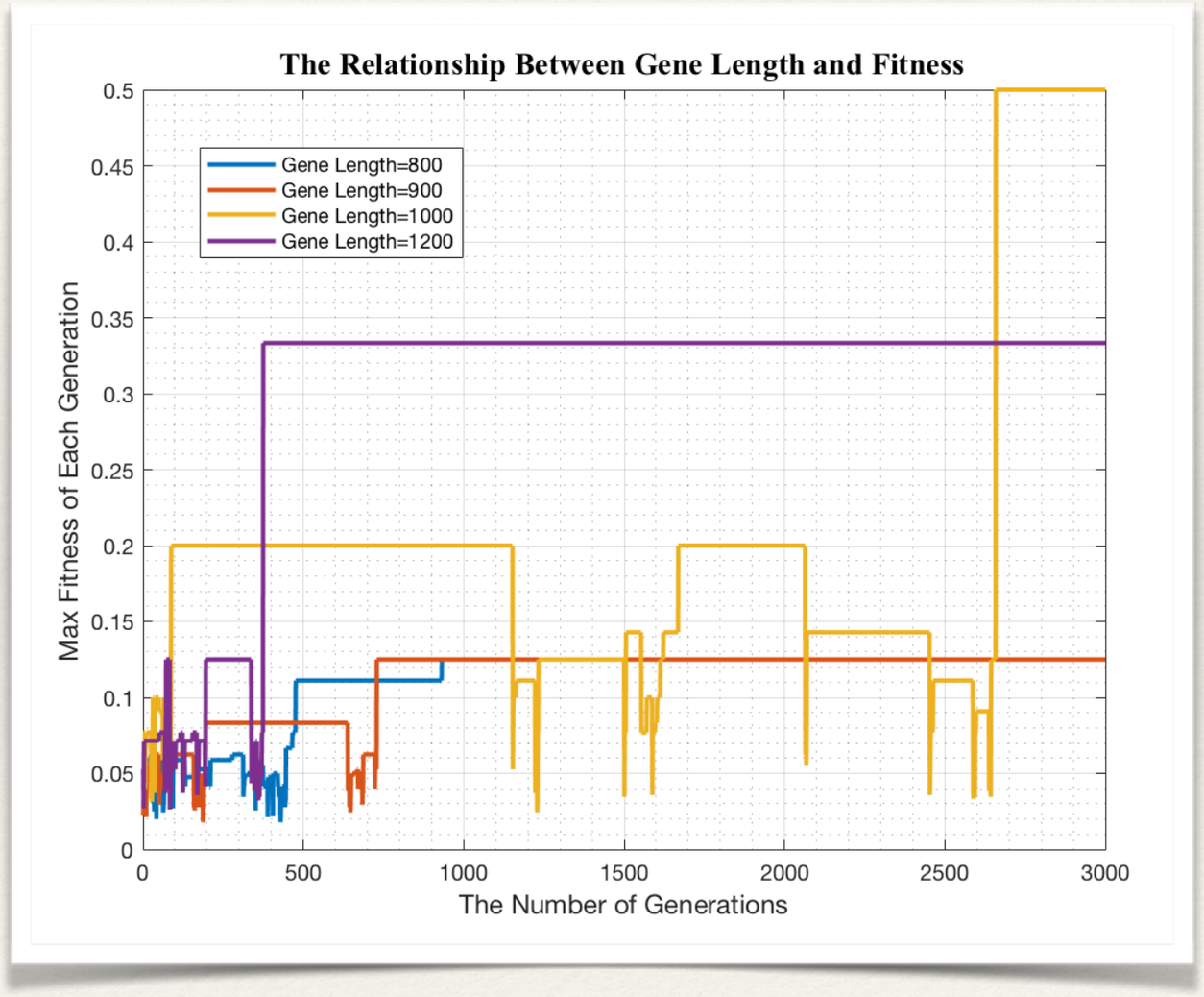
The Relationship Between Crossover_rate and Fitness Score

- When the crossover rate is too low, it will make the individual have the less chance to get chromosome both from their parents. Finally, it leads no individual gets to the exit within 3000 generations, it will need more generations to reach to the exit.
- When the crossover rate is high, especially 0.8 or higher, it will increase the diversity of chromosome, it leads individual to get to the exit easier.
- In conclusion, from the picture we can get that the suitable crossover rate is between 0.8-1.0. Finally, we select 0.9 as the crossover rate.

The Relationship Between Gene Length and Fitness Score

The picture shows that the relationship between different Gene Length and fitness scores within 3000 generations.

Because the Gene Length means the number of steps that each individual can take in the maze. The program stops only when the final step reaches the exit. So when the gene length equals to 800 and 900, the number of the steps is not enough to get the exit point. But if the gene length is too long, it will make the route repeated.

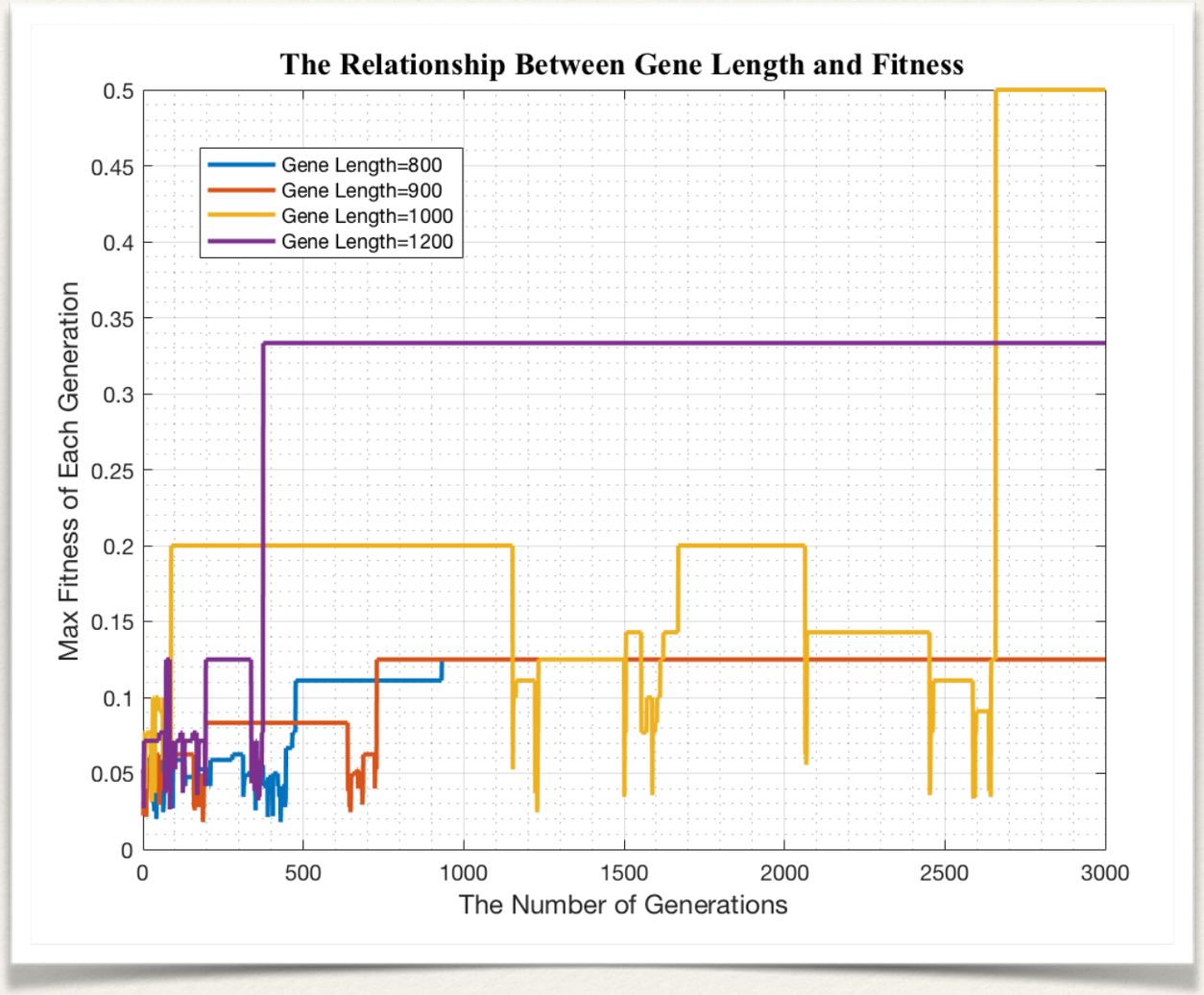


Crossover rate=0.9, Mutation rate=0.2

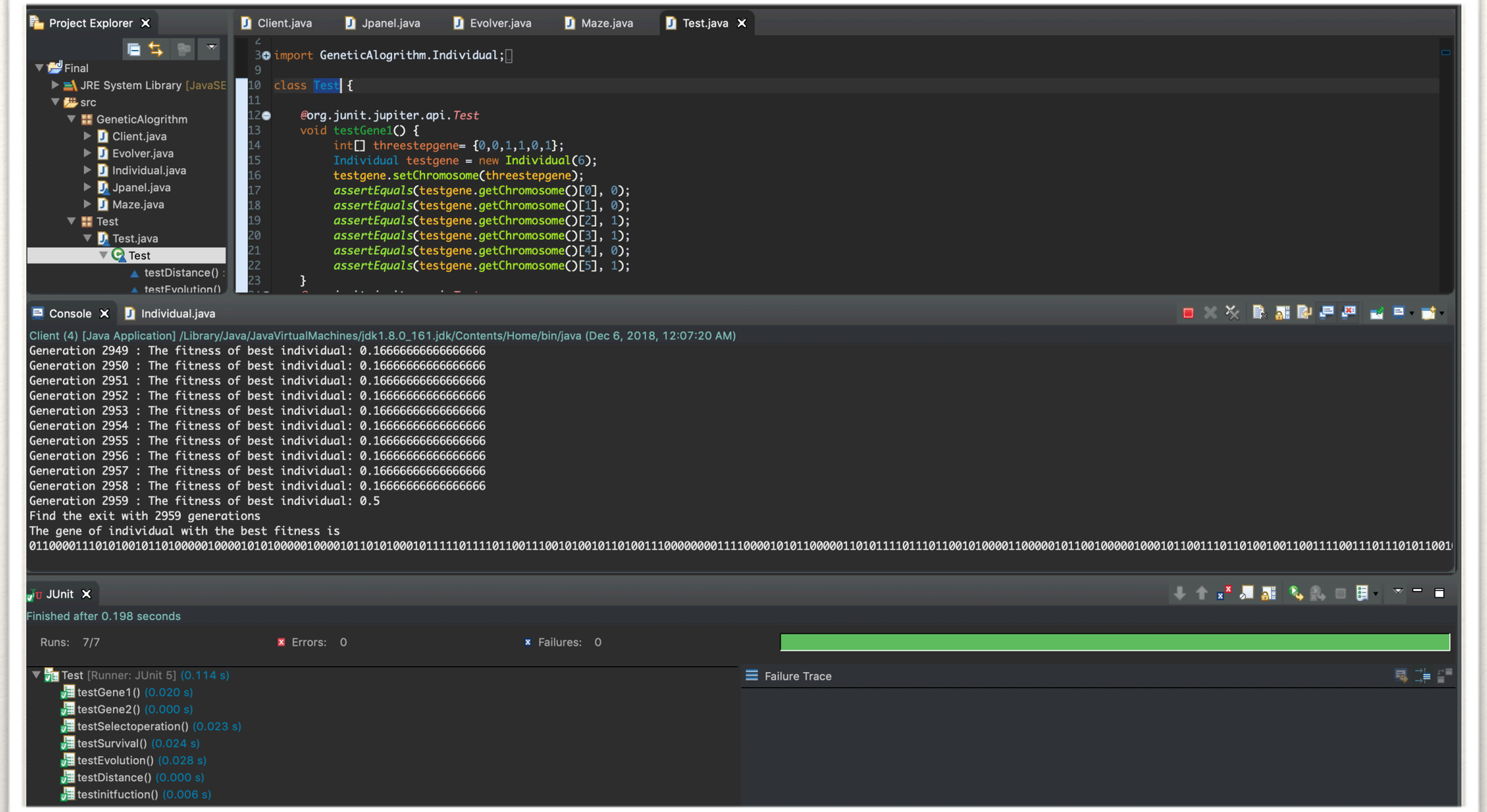
The Relationship Between Gene Length and Fitness Score

As shown in the picture, when the gene length equals to 1200, the max fitness score get high in 400 generations, and then keep the same score in the last of generations.

In a word, in the maze problem, we should select a suitable gene length, not too long or short. In this project, we chose Gene length equals to 1000.

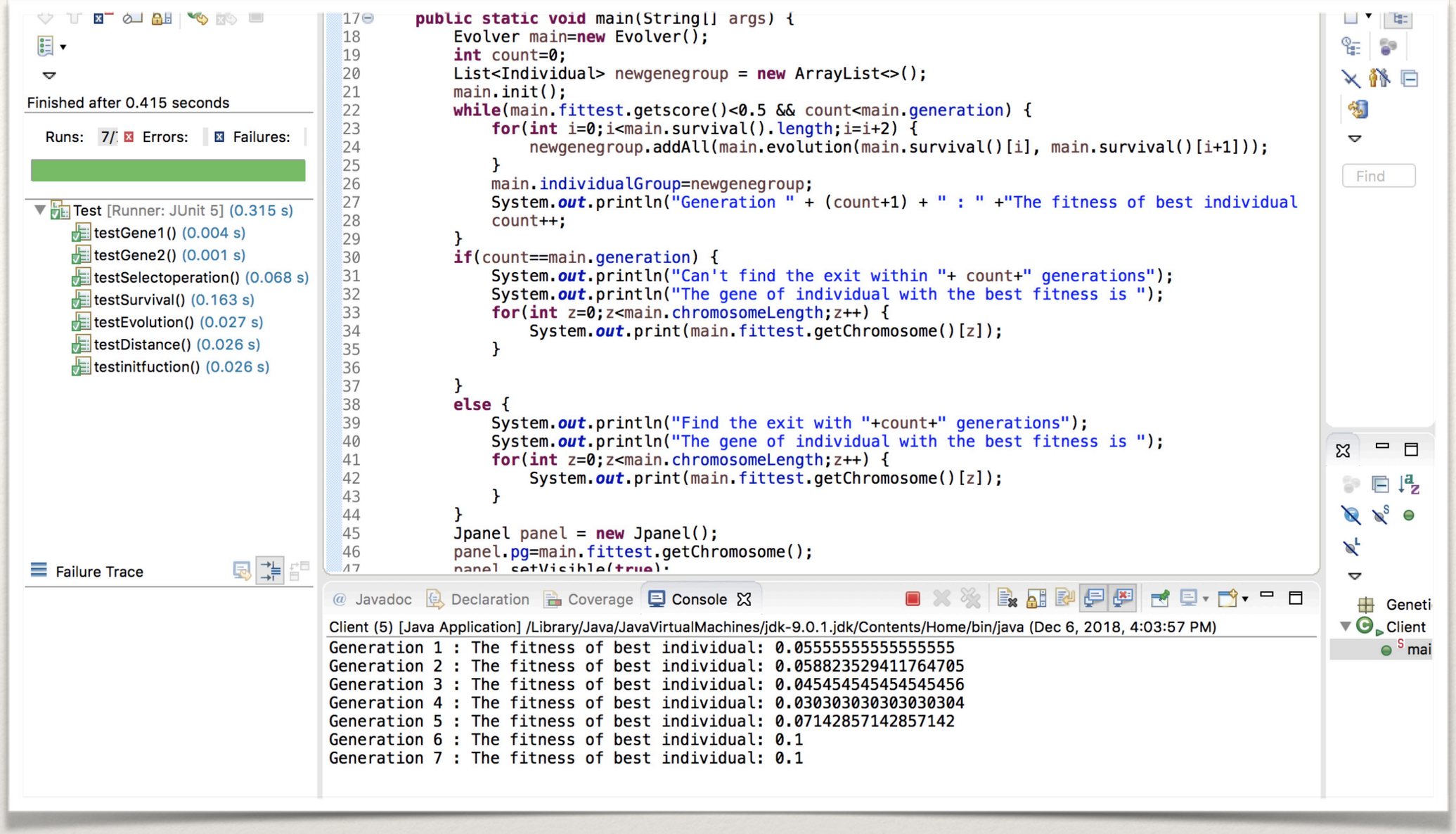


Crossover rate=0.9, Mutation rate=0.2



Results (screen shot)

The picture shows our unit test and client are running correctly, the client reach to exit at 2959 generations and output the gene with the best fitness.



Results (screen shot)

This picture shows the beginning of the test, you can see the individual with the best fitness is 0.055 in the first generation and increases to 0.1 in the sixth generation.

Results (Unit Tests)

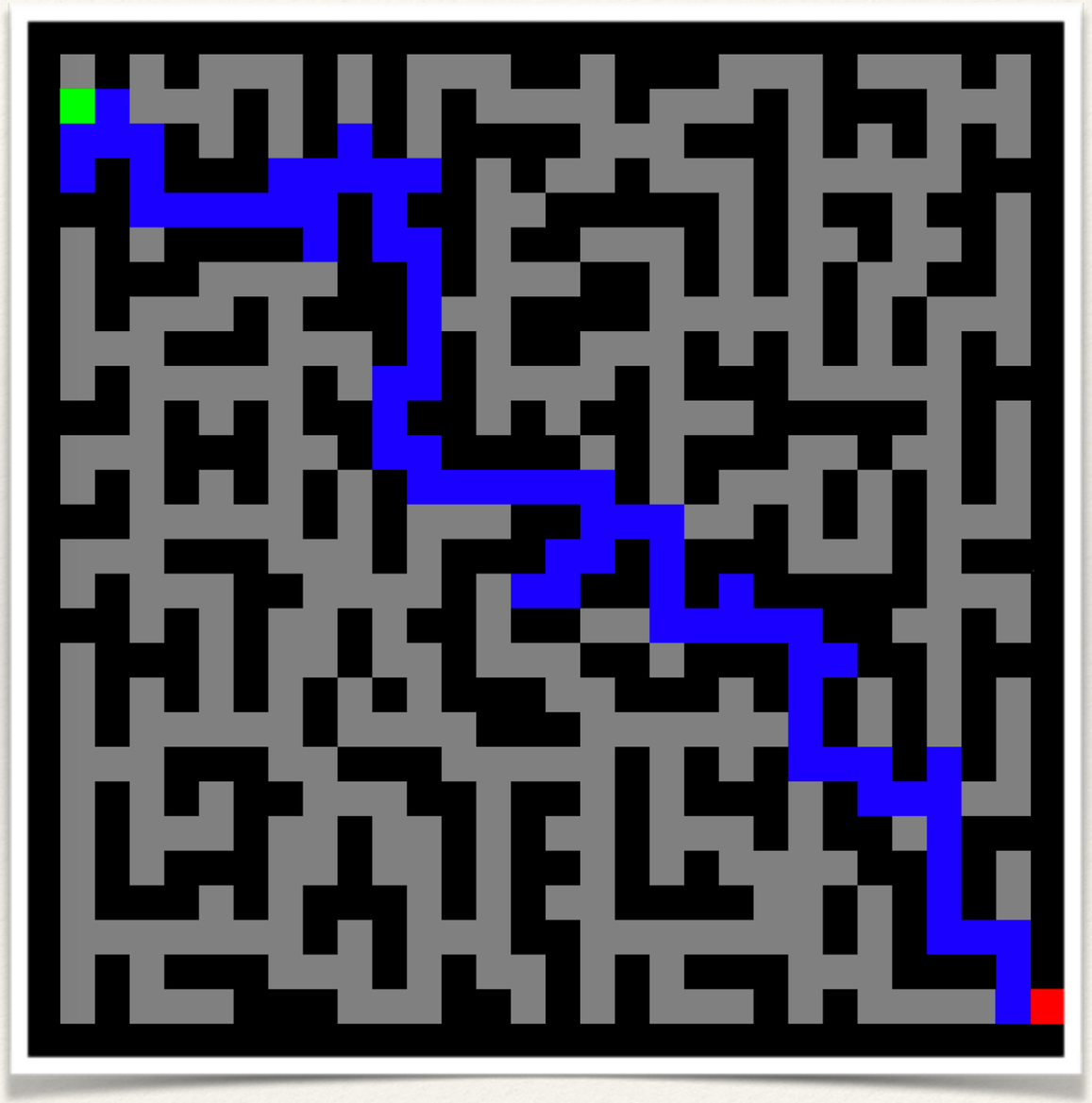
In this part, we wrote 7 unit tests to test different functions in Evolver Class(included Selection operation, Survival, and Evolution function) and Individual Class. All unit tests can be passed correctly.

```
@org.junit.jupiter.api.Test
void testGene1() {
     int[] threestepgene= {0,0,1,1,0,1};
     Individual testgene = new Individual(6);
     testgene.setChromosome(threestepgene);
     assertEquals(testgene.getChromosome()[0], 0);
     assertEquals(testgene.getChromosome()[1], 0);
     assertEquals(testgene.getChromosome()[2], 1);
     assertEquals(testgene.getChromosome()[3], 1);
     assertEquals(testgene.getChromosome()[4], 0);
     assertEquals(testgene.getChromosome()[5], 1);
}
@org.junit.jupiter.api.Test
void testGene2() {
     int[] threestepgene= {0,0,1,1,0,1,1,0,1,0};
     Individual testgene = new Individual(10);
     testgene.setChromosome(threestepgene);
     assertEquals(testgene.getChromosome()[0], 0);
     assertEquals(testgene.getChromosome()[2], 1);
     assertEquals(testgene.getChromosome()[4], 0);
     assertEquals(testgene.getChromosome()[6], 1);
     assertEquals(testgene.getChromosome()[8], 1);
}

@org.junit.jupiter.api.Test
void testinitfuction() {
    Evolver testinit =new Evolver();
    testinit.init();
     assertEquals(testinit.individualGroup.size(), testinit.individual_size);
     assertEquals(testinit.individualGroup.get(0).length, testinit.chromosomeLength);
     assertEquals(testinit.individualGroup.get(0).getscore(), 0,1);
}

@org.junit.jupiter.api.Test
void testDistance() {
    Evolver testcalculatescore =new Evolver();
    testcalculatescore.init();
    assertEquals(testcalculatescore.distance(testcalculatescore.individualGroup.get(0).chromosome),testcalculatescor
}

@org.junit.jupiter.api.Test
void testSelectoperation() {
    Evolver testDistance =new Evolver();
    testDistance.init();
    assertEquals(testDistance.selectOperation(testDistance.individualGroup),0, testDistance.individual_size);
}

@org.junit.jupiter.api.Test
void testSurvival() {
    Evolver testluckdog =new Evolver();
    testluckdog.init();
    assertEquals(testluckdog.survival().length,testluckdog.individual_size/2);
    Random random =new Random();
    assertEquals(testluckdog.survival()[random.nextInt(testluckdog.individual_size/2)],testluckdog.individual_size/2
}

@org.junit.jupiter.api.Test
void testEvolution() {
    Evolver testmate =new Evolver();
    testmate.init();
    assertEquals(testmate.evolution(0, 1).size(),4);
```

Results (Java Panel Interface)

To show the result directly, we used java panel interface to show the final route, as shown in picture.

Our program can find a route between the start(green block) and the end(red block) in the maze with the Genetic Algorithms(highlighted in blue).



Conclusions

- By using Genetic Algorithms, we could have more probability to select individual who has the better fitness to generate high-quality solution.(In this case, it's to find the fittest route of all individuals from entrance to exit in the maze)
- According to the maze map, to get the best optimal genetic algorithms model, we chose mutation rate=0.2, crossover rate=0.9, gene length=1000 as parameters. Based on multiple tests, with these parameters, There's a high probability of getting a result in 3000 generations.