INFO6205 Program Structure & Algorithms

GA Project

Xiaoxu Mu NUID: 001838943

Difu Chen NUID: 001856436

Jingxuan Li NUID: 001847157

1. Problem

Pac-man exists in a rectangular space of 10×10 grids. There are 40 beans and 10 bombs which are randomly placed in the 100 grids. When Pac-man is born, it randomly appears in an arbitrary square. And what it needs to do is to use its own strategy to eat beans. There are only 200 steps in total. If the Pac-Man eats a bean, it will get 10 points. If the Pac-Man eats a bomb, it will lose 3 points. If the Pac-Man knocks its head against a brick wall, it will lose 5 points. And it will also lose 1 point if it stops at a position in which there is no beans or bombs.

In theory, the highest score is to eat all the beans without deduction. There are 40 beans in total, which means in the ideal situation it can get 400 points.

According to the current state, Pac-Man decides to take the corresponding action. We use number 0 to 7 to respectively present the 8 actions, which are moving up, moving down, moving left, moving right, eating bombs, eating beans, staying and moving randomly.

The purpose of this problem is to find out a strategy to get the highest-score in this Pac-Man game. There are 8^1024 kinds of strategy combinations in total. If we want to calculate by exhaustive method, this problem is unsolvable. However, the genetic algorithm can solve this problem effectively.

2. Findings

The experimental steps of this study are as follows:

Step 1: Randomly generate 200 strategies, which is to generate 200 Pac-Men who use different strategies to eat beans.

Step 2: Each strategy carries out 1000 Challenges. Each game allowed the Pac-Man to use 200 steps. In the 1000 Challenges, 40 beans per game were randomly scattered. Finally, in the 1000 challenges, the average score is evaluated by every 200 steps.

Step 3: According to the level of the score, two strategies are randomly selected from the 200 strategies. It will choose the perfect strategies of higher sore. Then use the selected two strategies to generate a new strategy. Each column of the new strategy has half the probability to use the corresponding column of the first strategy, and half of the probability uses the corresponding column of the second strategy. In the process of generating a new strategy, there will be a small probability of generating a mutation of the strategy.

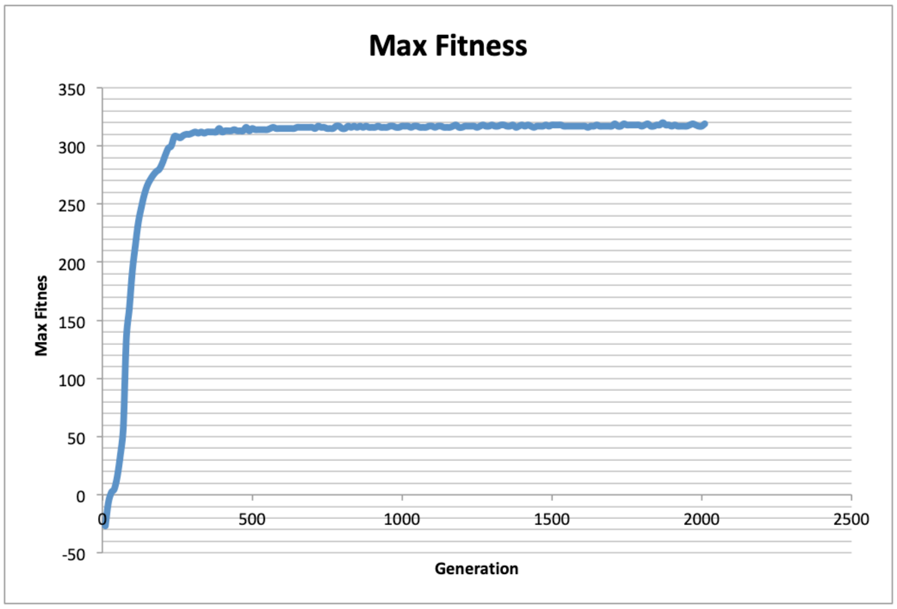
Step 4: Use the method of step 3 to continuously generate new strategies until 200 are generated;

Step 5: Go back to step 2, use the new generated strategy to play the game, and then generate a next-generation strategy based on the score.

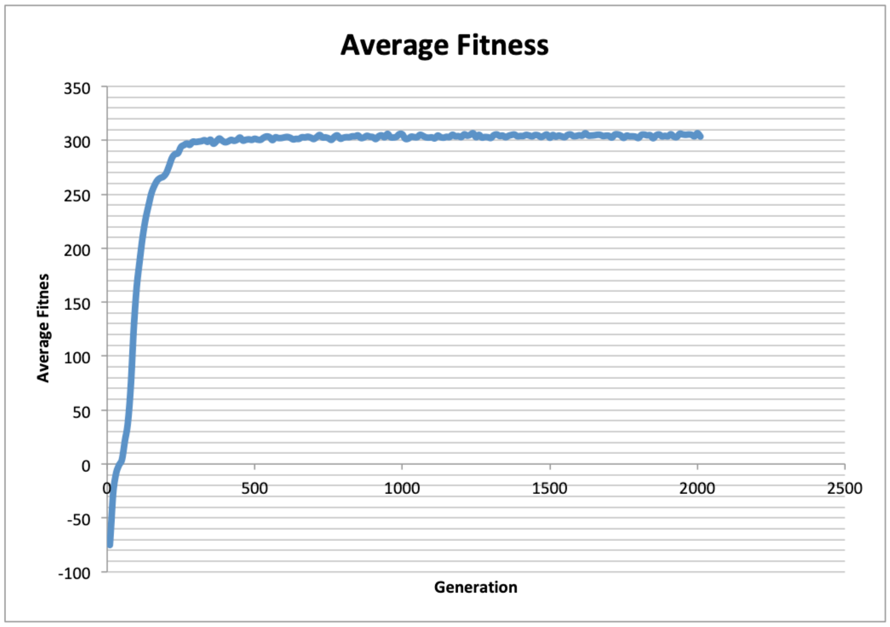
According to the above steps for 2000 cycles, there are 2000 generations of Pac-Man. The first generation of Pac-Man's strategy was randomly generated. The score could reflect whether the strategy is competitive. The experiment introduces the evolutionary behavior, treating each strategy in each Pac-Man strategy list as a gene. The first generation of higher-performing Pac-Man has a higher chance to leave descendant, and the offspring inherit the strategy of the previous generation. A low score also has a chance to generate offspring, but the probability is lower. In the process of generating the next generation, the next generation strategy has a probability of generating genetic mutation.

3. Results

There are 2000 generations of Pac-Man. The results of best fitness for each generation are as follows:



The results of average fitness of all the Pac-Man in each generation are as follows:



It can be seen that the first generation of strategies randomly generated Pac-Man is very poor, with a maximal fitness of -26 and a minimal fitness of -141. Because the strategy is randomly generated, there will always be a deduction of the wall, the deduction of the beans when there is no bean and a deduction of bomb. However, this situation has changed substantially in the 50th generation. From the 50th generation, the average fitness of the offspring exceeded 0. Since then, the fitness gradually increased. The 90th generation exceeded 100, and the 120th generation exceeded 200, and after 330th generation, the fitness exceeded 300, and finally rose to 306 in the 2000th generation.

4. Conclusions

By using genetic algorithm, we can solve this Pac-Man problem which has huge solution spaces effectively. It generates descendants with higher fitnesses. And as a result, models of 2000 generations can reach the highest fitness of around 306 in 200 steps.

5. Implementation Details

(1) There are 8 actions that the Pac-Man takes, which are moving up, moving down, moving left, moving right, eating bombs, eating beans, staying and moving randomly.In addition, Pacman can observe 5 positions:up,down,left,right and current grids, and each grid has four states: empty,wall, bean and bomb, so it can have 4^5 = 1024 states.

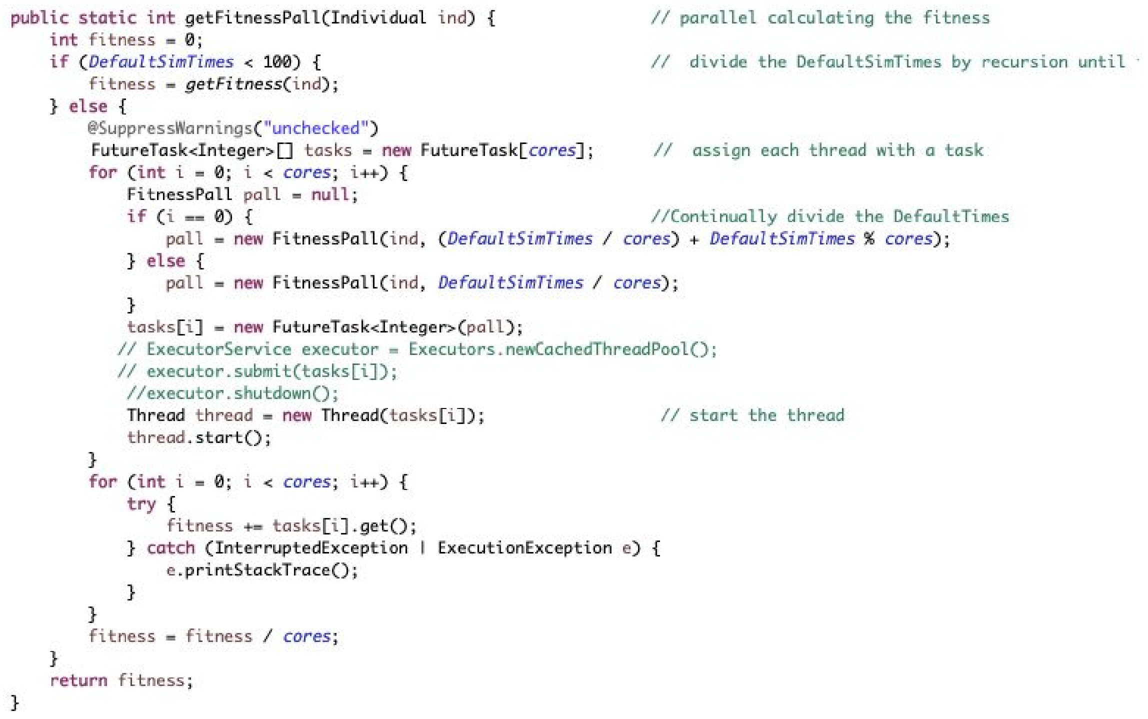
(2) Gene Expression: The genes of Pac-Man individuals can be represented by a 1024-length gene, corresponding to the 1024 states, and each gene has eight conditions, indicating the response produced in the state. We create a byte array to store genes and use generateGenes() method to generate individual genes randomly for each gene-position.

(3) The selection function: We set up a priority queue with a override comparator to select the individuals with higher fitnesses to generate descendants.

(4) The fitness function: Each individual can walk 200 steps. For each step,it can choose an action to perform.There is a counter for the fitness. If it choose to eat beans, the counter adds 10 points; if it choose to eat bombs, the counter will minus points; if it rushes the wall, the counter will minus 5 points; and if it eat empty, the counter will minus 1 point.

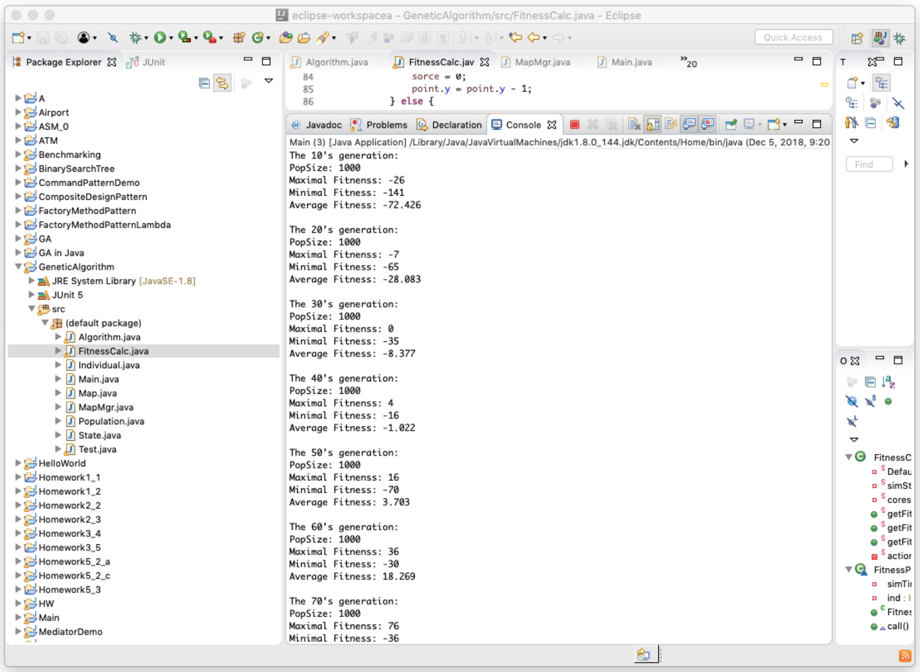
6. Evidence of parallelization

We used parallelization to parallel calculating the fitness.



```
public static int getFitnessPall(Individual ind) {                // parallel calculating the fitness
    int fitness = 0;
    if (DefaultSimTimes < 100) {                                  //  divide the DefaultSimTimes by recursion until
        fitness = getFitness(ind);
    } else {
        @SuppressWarnings("unchecked")
        FutureTask<Integer>[] tasks = new FutureTask[cores];      //  assign each thread with a task
        for (int i = 0; i < cores; i++) {
            FitnessPall pall = null;
            if (i == 0) {                                         //Continually divide the DefaultTimes
                pall = new FitnessPall(ind, (DefaultSimTimes / cores) + DefaultSimTimes % cores);
            } else {
                pall = new FitnessPall(ind, DefaultSimTimes / cores);
            }
            tasks[i] = new FutureTask<Integer>(pall);
           // ExecutorService executor = Executors.newCachedThreadPool();
           // executor.submit(tasks[i]);
            //executor.shutdown();
            Thread thread = new Thread(tasks[i]);                  // start the thread
            thread.start();
        }
        for (int i = 0; i < cores; i++) {
            try {
                fitness += tasks[i].get();
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
            }
        }
        fitness = fitness / cores;
    }
    return fitness;
}
```

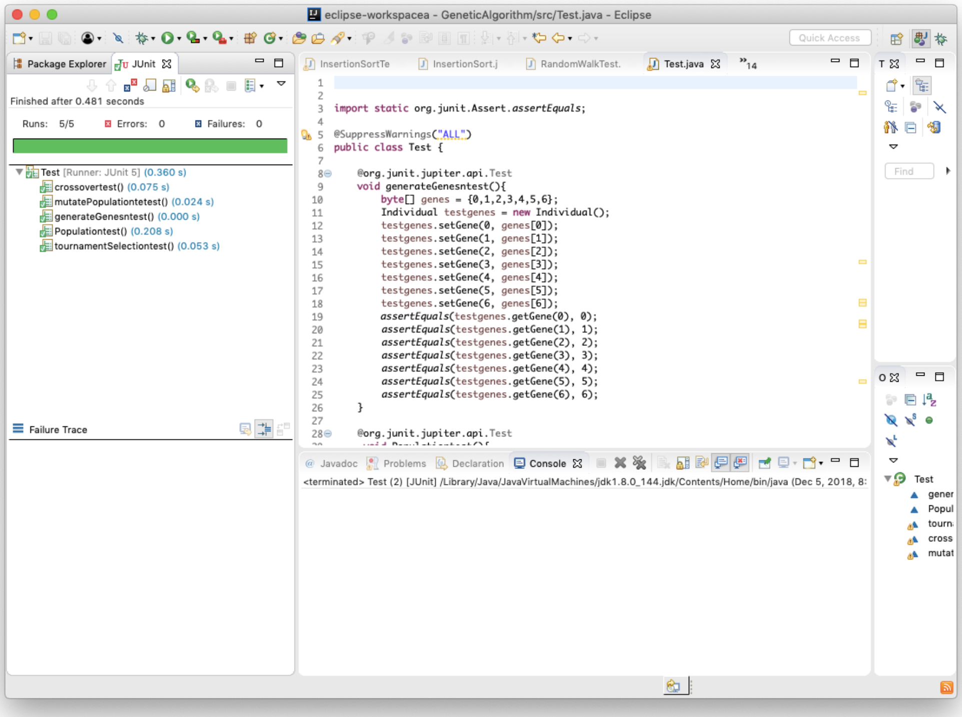
7. Evidence of your programming running



```
            sorce = 0;
            point.y = point.y - 1;
        } else {
```

```
Main (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (Dec 5, 2018, 9:20
The 10's generation:
PopSize: 1000
Maximal Fitnenss: -26
Minimal Fitness: -141
Average Fitness: -72.426

The 20's generation:
PopSize: 1000
Maximal Fitnenss: -7
Minimal Fitness: -65
Average Fitness: -28.083

The 30's generation:
PopSize: 1000
Maximal Fitnenss: 0
Minimal Fitness: -35
Average Fitness: -8.377

The 40's generation:
PopSize: 1000
Maximal Fitnenss: 4
Minimal Fitness: -16
Average Fitness: -1.022

The 50's generation:
PopSize: 1000
Maximal Fitnenss: 16
Minimal Fitness: -70
Average Fitness: 3.703

The 60's generation:
PopSize: 1000
Maximal Fitnenss: 36
Minimal Fitness: -30
Average Fitness: 18.269

The 70's generation:
PopSize: 1000
Maximal Fitnenss: 76
Minimal Fitness: -36
```

8. Screenshot of the unit tests all passing



```
Finished after 0.481 seconds

Runs: 5/5    Errors: 0    Failures: 0

Test [Runner: JUnit 5] (0.360 s)
    crossovertest() (0.075 s)
    mutatePopulationtetest() (0.024 s)
    generateGenesntest() (0.000 s)
    Populationtest() (0.208 s)
    tournamentSelectiontest() (0.053 s)
```

```java

import static org.junit.Assert.assertEquals;

@SuppressWarnings("ALL")
public class Test {

    @org.junit.jupiter.api.Test
    void generateGenesntest(){
        byte[] genes = {0,1,2,3,4,5,6};
        Individual testgenes = new Individual();
        testgenes.setGene(0, genes[0]);
        testgenes.setGene(1, genes[1]);
        testgenes.setGene(2, genes[2]);
        testgenes.setGene(3, genes[3]);
        testgenes.setGene(4, genes[4]);
        testgenes.setGene(5, genes[5]);
        testgenes.setGene(6, genes[6]);
        assertEquals(testgenes.getGene(0), 0);
        assertEquals(testgenes.getGene(1), 1);
        assertEquals(testgenes.getGene(2), 2);
        assertEquals(testgenes.getGene(3), 3);
        assertEquals(testgenes.getGene(4), 4);
        assertEquals(testgenes.getGene(5), 5);
        assertEquals(testgenes.getGene(6), 6);
    }

    @org.junit.jupiter.api.Test
```

```
<terminated> Test (2) [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (Dec 5, 2018, 8:
```