INFO 6205 Final Project Report

Mengfan Shi
Zhengyang Xu

Summary

For this project, our group chose Genetic Algorithms to play the Game of Life. For the initial start pattern, we use Jenetics library to randomly create the first generation pattern

```
public class InitPattern {
    public static void main(String[] args) {
        HashMap<String, String> pattern = getPattern();
        System.out.println(pattern);
    }
    public static HashMap<String, String> getPattern() {
        HashMap<String, String> startPointList = new HashMap<>();
        int count = 0;
        while (count < 8) {
            int numberOfPoint = 8;
            Genotype<BitGene> gtf
                    = Genotype.of(BitChromosome.of(8, 0.05), numberOfPoint);
```

which is a set of bits such as
[00011000,10001010,00000000,00000001,00001110,01100000,01100000,00010110].

These pattern bits can be translated into a set of points, in this case the matrix should be like

□□□■■□□□□□
■□□□■□□■□□□
□□□□□□□□□□
□□□□□□□□■□□
□□□□■■■□□□
□■■□□□□□□□
□■■□□□□□□□
□□□■□□■■□□□
□□□□□□□□□□
□□□□□□□□□□

Then the above points are transferred into eight strings using StringBuilder(). Details can see the InitPattern class.

```
StringBuilder stringBuilder = new StringBuilder();
for (int i = 0; i < gtf.length(); i++) {
    Chromosome<BitGene> bitGenes = gtf.get(i);
    String bit = bitGenes.toString();
    StringBuilder line = new StringBuilder();
    for (int j = 0; j < bit.length(); j++) {
        Character c = bit.charAt(j);
        if (c.equals('1')) {
            line.append(i);
            line.append(" ");
            line.append(j);
            line.append(",");
        }
    }
```

For the GameStart class, the obtained eight strings should be passed into the method:

```
Game.Behavior run = Game.run(0L, b);
```

By using run.Generation method, we can get the life cycle (generation) of each passed pattern. The survival() method which in our case is the seleceTopPattern() method can be used to select the pattern who has the longest life cycle (generation).

```
private static List<String> selectTopPattern() {
    List<Map.Entry<String, Long>> list = new ArrayList<>(saveGenerationCount.entrySet());
    list.sort((o1, o2) -> (int) (o2.getValue() - o1.getValue()));
    list.forEach(a -> System.out.println(a.getValue() + "  " + a.getKey()));
    List<Map.Entry<String, Long>> subList = list.subList(0, 1);
    return subList.stream().map(Map.Entry::getKey).collect(Collectors.toList());
}
```

If the life cycle > 1000, then this pattern is the fittest, output the result, otherwise the pattern should start to mutate, referring to mutation() method until the life cycle is > 1000.

```
<terminated> GameStart [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.jdk/Contents/Home/bin/java (2019
Group generation: 994
generation 995; grid=Grid{generation=995, groups=[generation 995, origin = {1, 1}, exter
    [{0, 0}, {0, 1}, {-1, 1}, {-2, 0}, {-1, 2}, {-236, -248}, {-237, -249}, {-235, -248}
generation 995;
count=12
Group generation: 995
generation 996; grid=Grid{generation=996, groups=[generation 996, origin = {1, 1}, exter
    [{0, 0}, {0, 1}, {0, 2}, {-1, 2}, {-2, 1}, {-236, -248}, {-237, -249}, {-235, -248},
generation 996;
count=12
Group generation: 996
generation 997; grid=Grid{generation=997, groups=[generation 997, origin = {1, 1}, exter
    [{0, 0}, {-1, 0}, {-2, -1}, {-1, 1}, {-2, 1}, {-237, -249}, {-238, -250}, {-236, -24
generation 997;
count=12
Group generation: 997
generation 998; grid=Grid{generation=998, groups=[generation 998, origin = {1, 1}, exter
    [{0, 0}, {-1, -1}, {0, 1}, {-1, 1}, {-2, 1}, {-237, -249}, {-238, -250}, {-236, -249
generation 998;
count=12
Group generation: 998
generation 999; grid=Grid{generation=999, groups=[generation 999, origin = {1, 1}, exter
    [{0, 0}, {0, 1}, {-1, 1}, {-2, 0}, {-1, 2}, {-237, -249}, {-238, -250}, {-236, -249}
generation 999;
count=12
Group generation: 999
Terminating due to: having exceeded 1000 generations
```

Initial pattern:

□□□■■□□□□□

■□□□■□□■□□□

□□□□□□□□□□

□□□□□□□■□□

□□□□■■■□□□

□■■□□□□□□□

□■■□□□□□□□

□□□■□■■□□□
□□□□□□□□□□
□□□□□□□□□□

First grade pattern:

□□□■□■□□□□
□□□■□■□□□□
□□□□□□■□□□
□□□□□■■□□□
□□■□□□□□□□
□■□■□■■□□□
□■□■■□□□□□
□□■□□□□□□□
□□□□□□□□□□
□□□□□□□□□□

Second grade pattern:

□□□□□□□□□□
□□□□□■■□□□
□□□□■□□■□□□
□□□□□■■□□□
□□■□□■□□□□□
□■□■□□■□□□□
□■□■■■□□□□
□□■■□□□□□□
□□□□□□□□□□
□□□□□□□□□□

Third grade pattern:

□□□□□□□□□□
□□□□□■■□□□
□□□□■□□□■□□
□□□■■□□■□□□
□□■■■□□■□□□
□■□□□■□□□□
□■□□□■□□□□
□□■■□□□□□□
□□□□□□□□□□
□□□□□□□□□□

Fourth grade pattern:

□□□□□□□□□□
□□□□□■■□□□
□□□■■□□□■□□
□□■□□□■■□□
□□■□□□■□□□

□■□■□■■□□□
□■□□■□□□□□
□□■□□□□□□□
□□□□□□□□□□
□□□□□□□□□□

Fifth grade pattern:
□□□□□□□□□□
□□□□■■■□□□
□□□■■□□■□□
□□■□□■■■□□
□■■■□□□□□□
□■□■■■■□□□
□■□■■■□□□□
□□□□□□□□□□
□□□□□□□□□□
□□□□□□□□□□

Methods

1) Genotype:

[00011000,10001010,00000000,00000001,00001110,01100000,01100000,00010110

2) Phenotype:

0 3,0 4,1 0,1 4,1 6,3 7,4 4,4 5,4 6,5 1,5 2,6 1,6 2,7 3,7 5,7 6

3) Fitness:

```
public static boolean fitness(String pattern) {
    return Game.run(0L, pattern).generation >= 1000;
}
```

4) Survival:

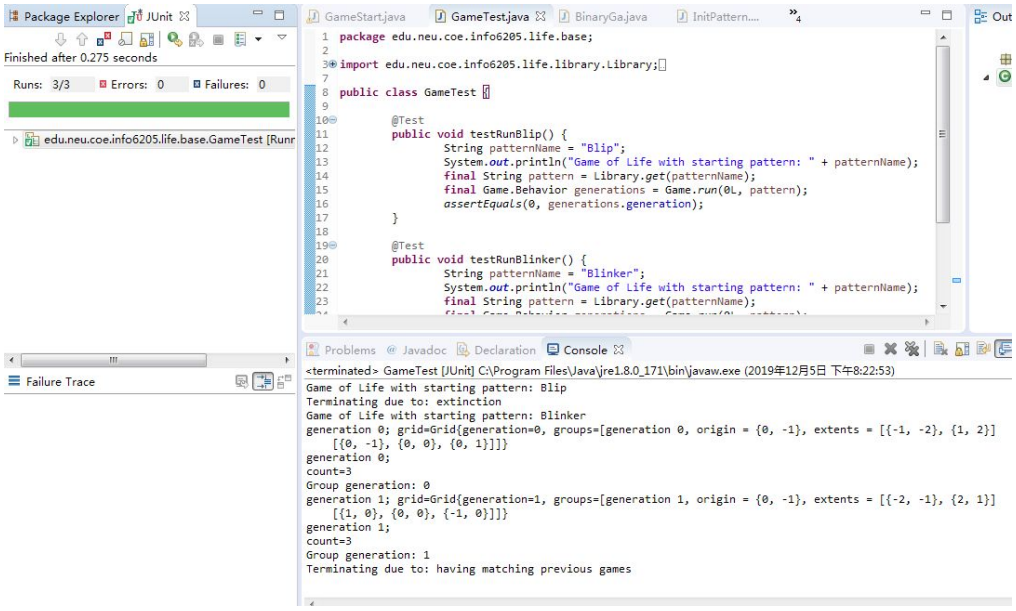
```
private static List<String> selectTopPattern() {
    List<Map.Entry<String, Long>> list = new ArrayList<>(saveGenerationCount.entrySet());
    list.sort((o1, o2) -> (int) (o2.getValue() - o1.getValue()));
    list.forEach(a -> System.out.println(a.getValue() + "  " + a.getKey()));
    List<Map.Entry<String, Long>> subList = list.subList(0, 1);
    return subList.stream().map(Map.Entry::getKey).collect(Collectors.toList());
}
```

5) Mutate:

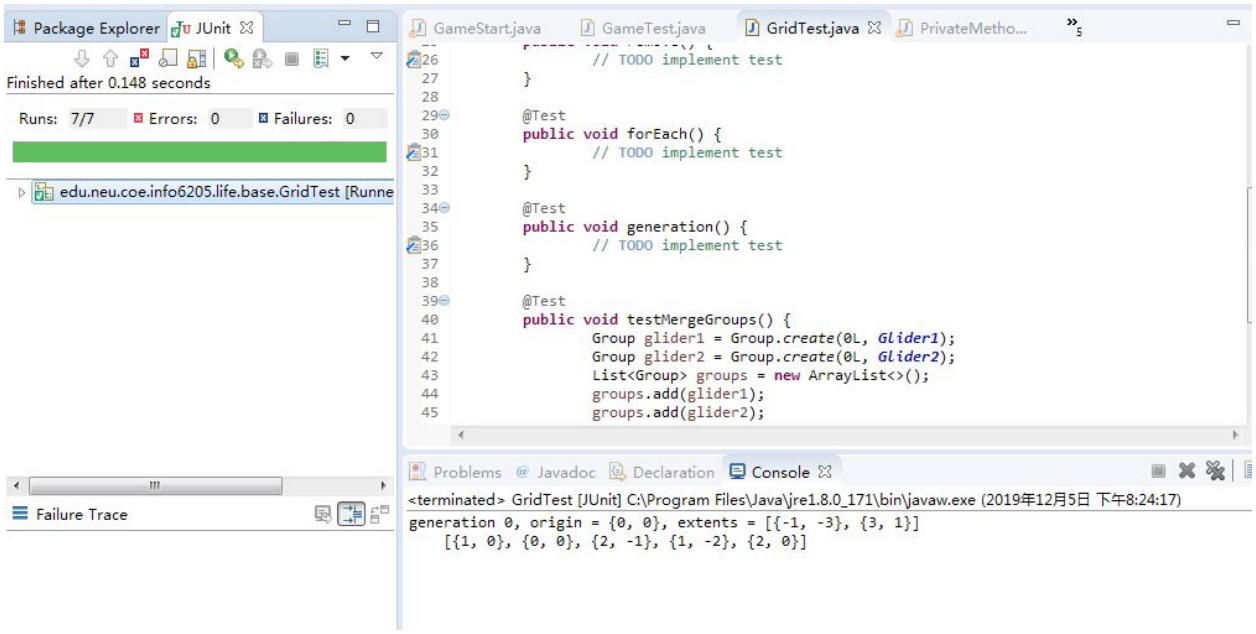
```
public static int[] swapMutation(int[] parent){
    int[] array = parent.clone();
    int l = array.length;
    //get 2 random integers between 0 and the size of the array
    int r1 = randomNumber(0,l);
    int r2 = randomNumber(0,l);
    //to make sure that two numbers are different
    while(r1 == r2) r2 = randomNumber(0,l);

    //swap array elements at those indices
    int temp = array[r1];
    array[r1] = array[r2];
    array[r2] = temp;

    return array;
}
```

Unit Tests

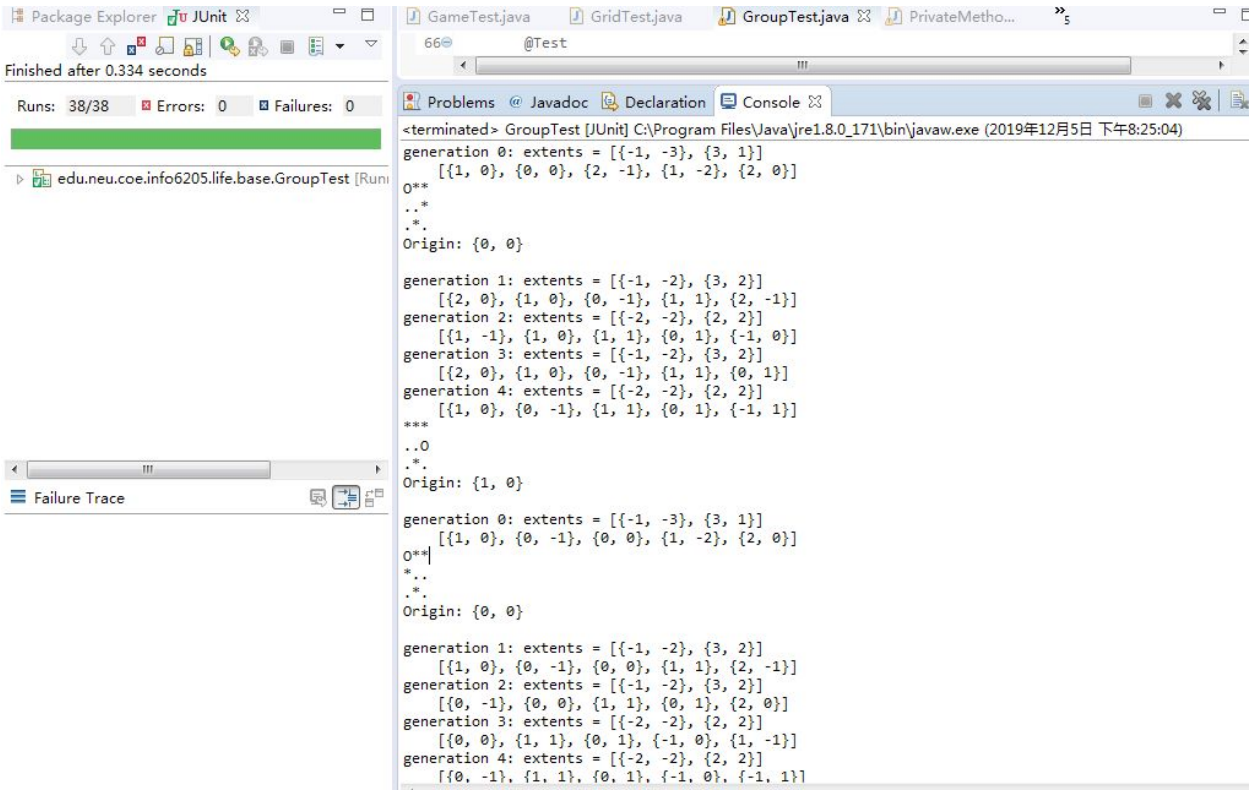
1) GameTest



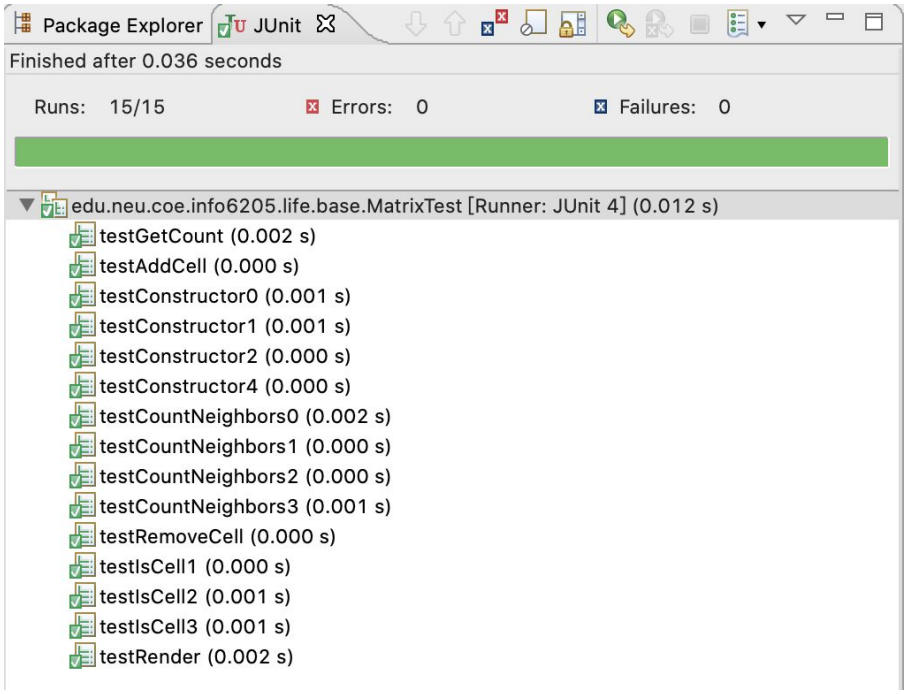
2) GridTest



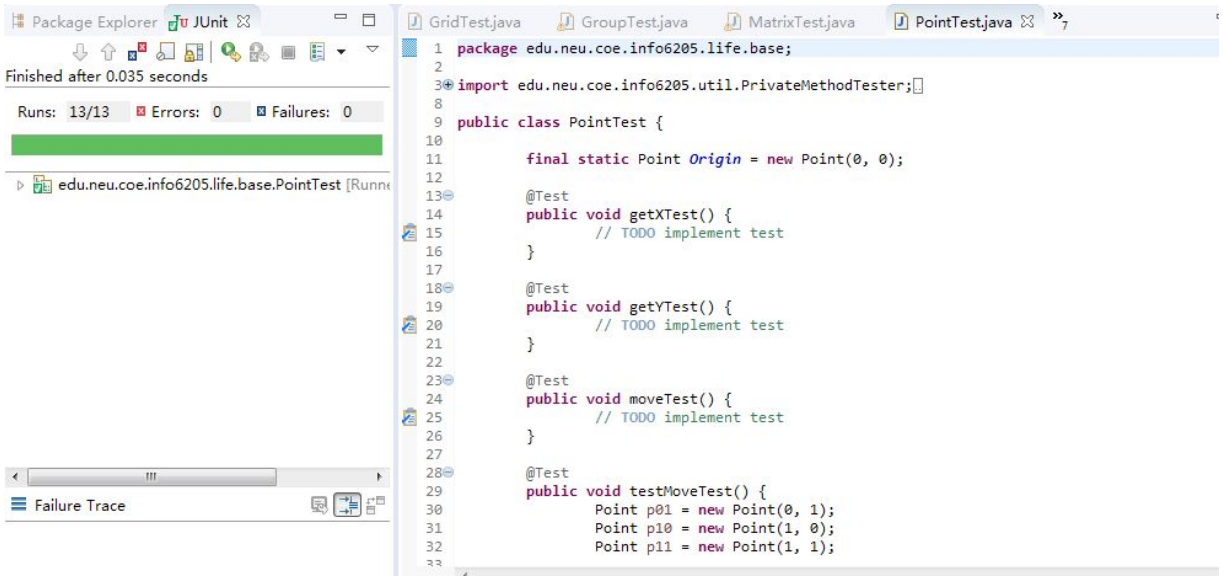
3) GroupTest



4) Matrix Test



5) PointTest



6) PrivateMethodTester