Program Structures & Algorithms

INFO 6205 Fall 2019

Final Project: Genetic Algorithms

Team Number: 113

Team member: Shinan Liu (001439919)

Zonghan Wu (001446435)

1. Abstract

In this project, a Genetic algorithm is used to find the best pattern of the Genotype in such an environment where Genotype can have the mutation and the "environment selection" configured by the program will choose one of those patterns which can procedure enough generation.

2. Implementation Details

1) **Genotype:** Our genotype only has one chromosome, which was made of a string of random integers whose range is from 6 to 500, and the length of the genotype must be an even number. In this way, the genotypes are totally different when the user runs the program and it can have a better stimulation effect.

2) **Genotype Decode:** In our case, the genotype will be covered to a serial of point coordinates as the initial input pattern for the Game. Because we have function to make sure the length of the Genotype is even, which is length N divide by two is zero, so that the N can be decode to N/2 pair of two integers, represent a pattern which has N/2 cells, eg: the random genotype is 123349, so we got three cells in our pattern, which are (1, 2), (3, 3), and (4, 9).

3) **Phenotype:** After decoding the genotype to a pattern of Game of life, our phenotype is a Game of life game that starts from that certain pattern.

4) **The fitness function:** Each *game* has a fitness attribute to judge whether the genotype achieves the preconfigured goal and how the genotype is close to the goal. We decode our genotype to phenotype, which is a game with a certain pattern, we use the generation of that game as the fitness value. A game will get more fitness scores along with more generation the game last.

5) **Goal:** The goal was set to finding the first phenotype(the game generation in our case) which has at least 100 generations. If the driver() cannot find the genotype which can complete the object, it will return the best pattern it has hitherto.

6) **Alterer:** Our goal is to find a pattern that has more generations, we use elitism to make sure good patterns will alive to the next evolution generation. To accelerate the search speed we use mutation to keep the diversity of our population. We do not use the crossover to get higher stability in the population.

7) **The survival function:** The mutation rate is *0.02%* and the preconfigured population size is 200. Then the pattern will follow the basic rules of the game of life.

8) **The evolution driver**: HelloWorld is the driver of this application. The driver simply instantiates a new evolution process with a set of values for the population size, elitism ratio, and mutation ratio, as well as a maximum number of generations to create before exiting the simulation, in order to prevent a potential infinite execution.

9) **Observation of the evolution:** The population is sorted by the fitness, so the best candidate from the final generation will be conveyed to the UI section to be displayed.

3. Architecture

I. Genetic Algorithm

1) Population

Our population is set to 200 in each generation of the evolution process.

The Population has 2 key attributes (an elitism ratio, and a mutation ratio), along with a collection of Chromosome instances, up to pre-defined population size. There is also an evolve() function that is used to evaluate the members of the population.

2) Evolution

First, we get the length of the chromosome by our random function. And each gene on the chromosome is random between 0 to 100. So that every point $P_{(x, y)}$ ($x \subset [0, 100], y \subset [0, 100]$)

Then we create 200 individuals as our first-generation individual, in our case, which are 200 Games. We run the games and get their game generation, calculate their fitness value. According to their fitness value, the elitism ratio is used to copy over a certain number of individuals unchanged to the new generation who have the best score. The remaining individual is then either muted dead or copied over directly, depending on the engine setting.

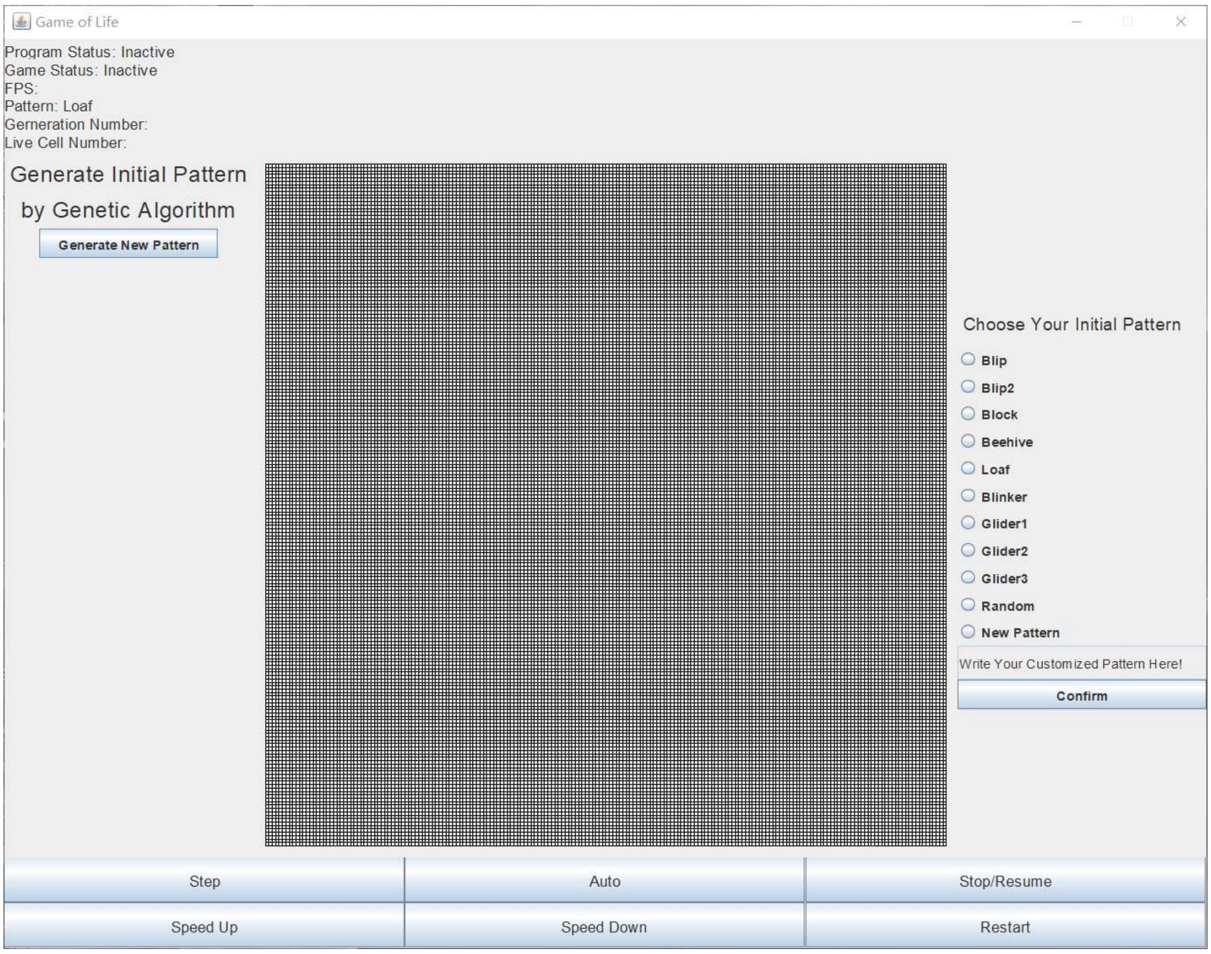
II. Evolution Truncation

In order to prevent a potential infinite execution, the evolution process will be truncate once it meets one of the two conditions.

In the evolution process, we had got a pattern that can let the game last longer than 100 generations. In the evolution process we haven't found a pattern match our goal, and our population evolution to 10000 generations, the process will be terminated at once.

III. UI

The UI has six components: Main Frame, Operation Panel, Grid Panel, Radio Button Panel, Label Panel, and Configuration Panel.



Program Status: Inactive
Game Status: Inactive
FPS:
Pattern: Loaf
Gerneration Number:
Live Cell Number:

Fig. 1 UI

1) **Main Frame:** Main Frame is used to contain the rest of the components and some of the important parameters like controlling the size of components are loaded when the user calls this frame. Main Frame uses the Border Layout to manage the other five panels.

2) **Grid Panel**: This panel is used to display the pattern, to manage how to paint the new pictures and to run the game of life with the basic rules. Points List is one way of confirming the initial pattern by converted as a 2D Boolean array which has the same size as the Grid. For instance, if the value of one "point" in this array is false, this cell is dead. The basic rules are implemented by using the absolute coordinates to compute whether the cell is alive or not in the next generation. However, the rules used int the program still have slightly different between the classical the game of life, which can lead to some interesting "outcomes" that will be mentioned in the next section.

3) **Operation Panel**: This panel can control the way of illustrating the pictures drawn by the Grid Panel. It has six functions: draw the picture step by step or automatically, accelerate or slow down the speed of drawing the picture, pause this game and restart to the initial pattern of your choices.

4) **Label Panel**: This panel's function is to show some parameters of the current game and it will be updated with the transformation of pictures and initial patterns.

5) **Radio Button Panel**: This component can let users choose the initial pattern from the built-in library, from what they create and even choose the totally random map as the initial pattern.

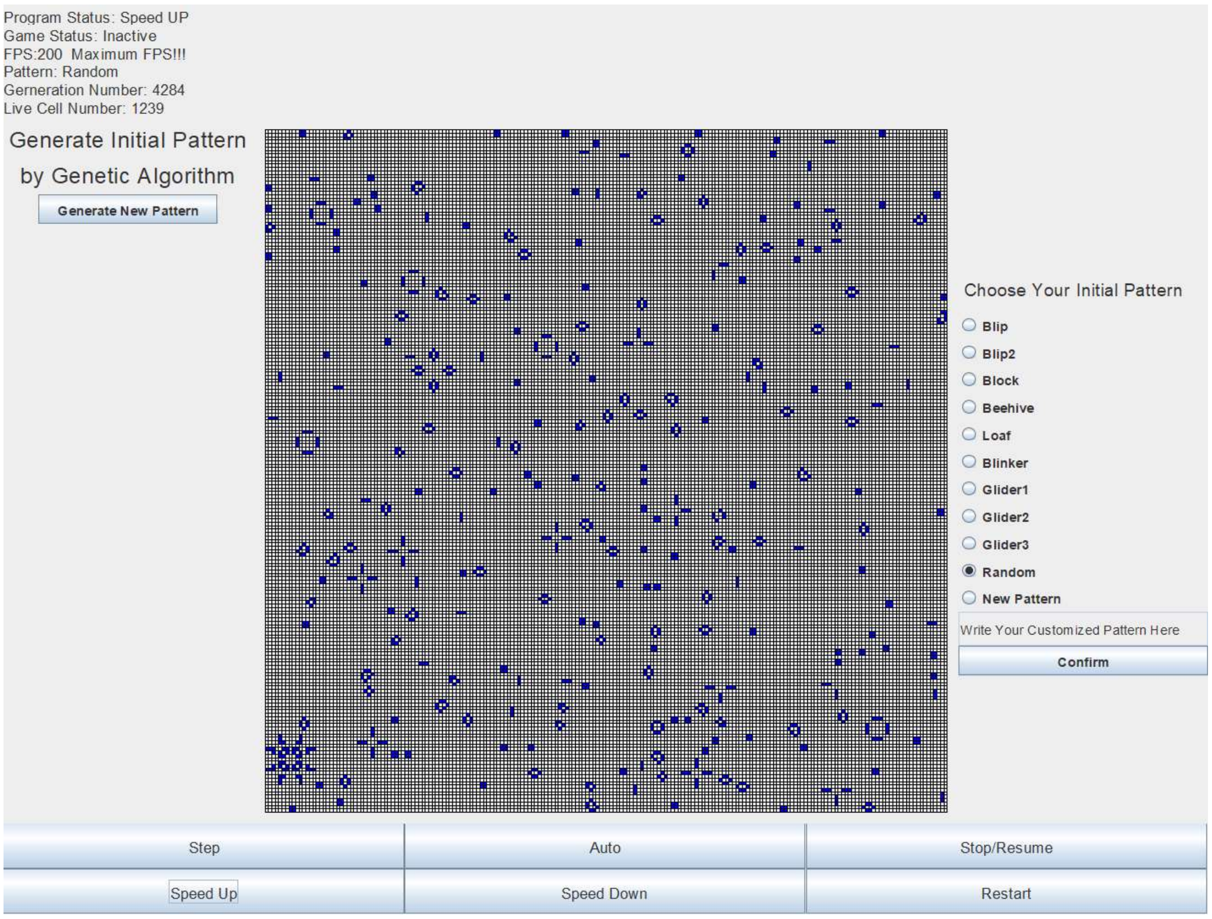
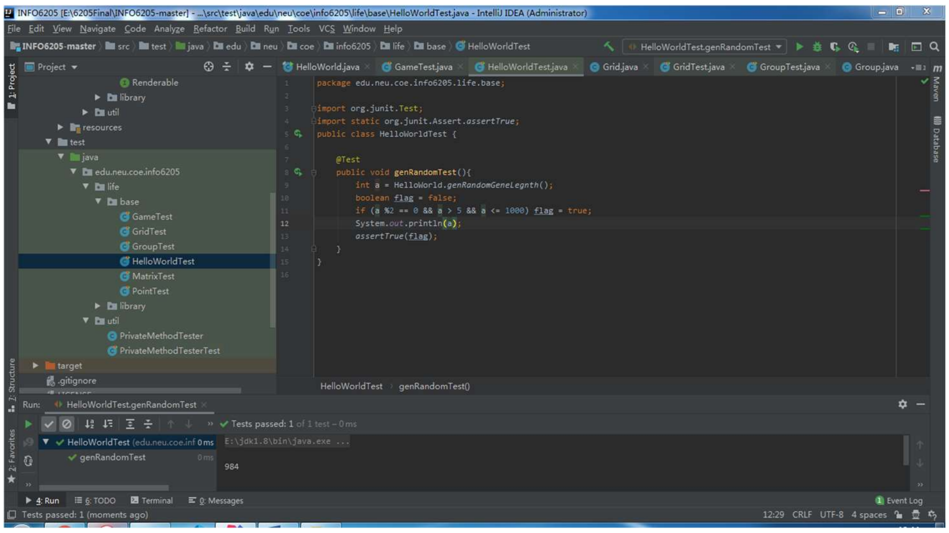

Program Status: Speed UP
Game Status: Inactive
FPS:200 Maximum FPS!!!
Pattern: Random
Gerneration Number: 4284
Live Cell Number: 1239

Fig. 2 A Static Game Created by Random Mode

6) Configuration Panel

This panel can let the pattern created by GA be the initial pattern for the Game. Besides, this panel reserves enough space for the buttons which have more functions in the future. In the original plan, the buttons on this panel can even change the size of the grid, the color of small blocks, or even modify some parameters of GA or the rules of the game of life. However, because of the limits of time, these functions have to be added in the near future.

IV. Unit Test



INFO6205 [E:\6205Final\INFO6205-master] - ...\src\test\java\edu\neu\coe\info6205\life\base\HelloWorldTest.java - IntelliJ IDEA (Administrator)

```java
package edu.neu.coe.info6205.life.base;

import org.junit.Test;
import static org.junit.Assert.assertTrue;
public class HelloWorldTest {

    @Test
    public void genRandomTest(){
        int a = HelloWorld.genRandomGeneLegnth();
        boolean flag = false;
        if (a %2 == 0 && a > 5 && a <= 1000) flag = true;
        System.out.println(a);
        assertTrue(flag);
    }
}
```

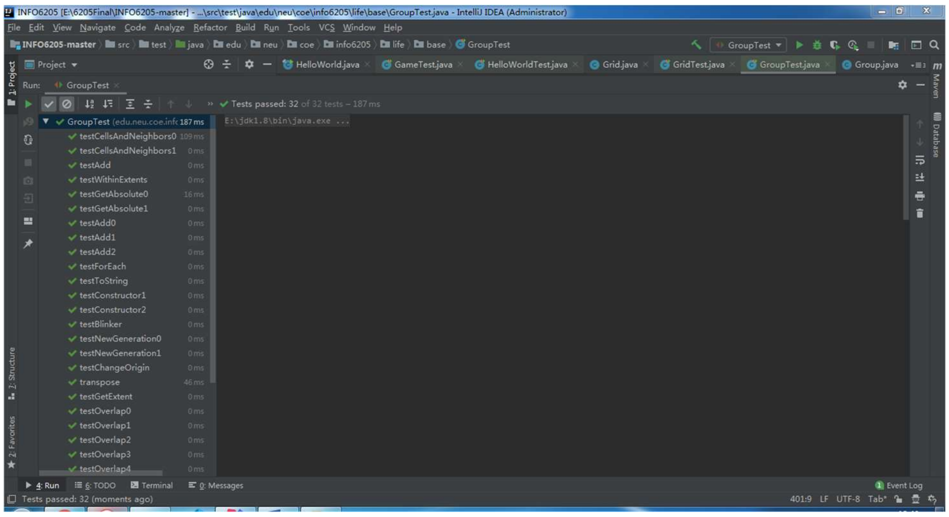
Run: HelloWorldTest.genRandomTest

Tests passed: 1 of 1 test – 0 ms

```
HelloWorldTest (edu.neu.coe.inf 0 ms
    genRandomTest                0 ms
```

```
E:\jdk1.8\bin\java.exe ...

984
```

Tests passed: 1 (moments ago)



INFO6205 [E:\6205Final\INFO6205-master] - ...\src\test\java\edu\neu\coe\info6205\life\base\GroupTest.java - IntelliJ IDEA (Administrator)

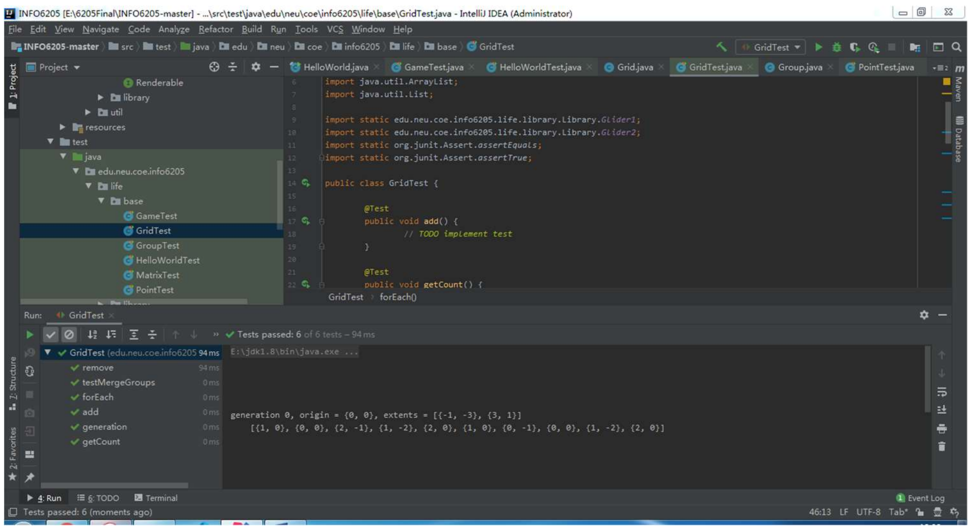
Run: GroupTest

Tests passed: 32 of 32 tests – 187 ms

```
GroupTest (edu.neu.coe.infc 187 ms
    testCellsAndNeighbors0   109 ms
    testCellsAndNeighbors1     0 ms
    testAdd                    0 ms
    testWithinExtents          0 ms
    testGetAbsolute0          16 ms
    testGetAbsolute1           0 ms
    testAdd0                   0 ms
    testAdd1                   0 ms
    testAdd2                   0 ms
    testForEach                0 ms
    testToString               0 ms
    testConstructor1           0 ms
    testConstructor2           0 ms
    testBlinker                0 ms
    testNewGeneration0         0 ms
    testNewGeneration1         0 ms
    testChangeOrigin           0 ms
    transpose                 46 ms
    testGetExtent              0 ms
    testOverlap0               0 ms
    testOverlap1               0 ms
    testOverlap2               0 ms
    testOverlap3               0 ms
    testOverlap4               0 ms
```

```
E:\jdk1.8\bin\java.exe ...
```

Tests passed: 32 (moments ago)



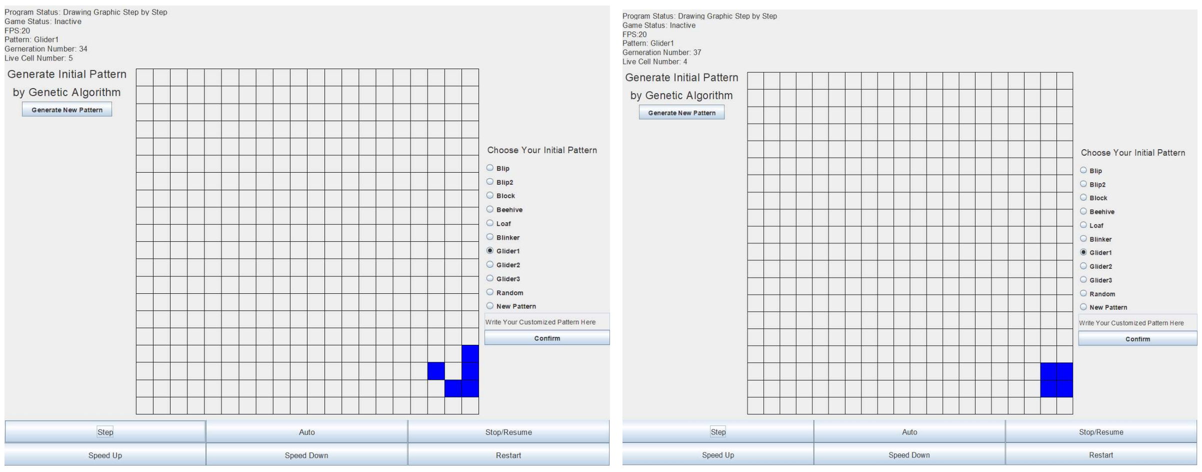
```
import java.util.ArrayList;
import java.util.List;

import static edu.neu.coe.info6205.life.library.Library.Glider1;
import static edu.neu.coe.info6205.life.library.Library.Glider2;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class GridTest {

        @Test
        public void add() {
                // TODO implement test
        }

        @Test
        public void getCount() {
```

```
Tests passed: 6 of 6 tests – 94 ms

GridTest (edu.neu.coe.info6205 94 ms
    remove            94 ms
    testMergeGroups    0 ms
    forEach            0 ms
    add                0 ms
    generation         0 ms
    getCount           0 ms

E:\jdk1.8\bin\java.exe ...

generation 0, origin = {0, 0}, extents = [{-1, -3}, {3, 1}]
    [{1, 0}, {0, 0}, {2, -1}, {1, -2}, {2, 0}, {1, 0}, {0, -1}, {0, 0}, {1, -2}, {2, 0}]

Tests passed: 6 (moments ago)
```

4. Problem

The pattern created by the GA truly can last enough generations in the Grid Panel as it did in the driver, however, some of the patterns didn’t have the same effect. There may be two possible reasons for this phenomenon.

1) The slight difference between the basic rules loaded in the Grid Panel and the classical game of life. For example, in the classical type, some patterns like Glider have a movable repeated pattern, but in the rules of Grid Panel, it will become a pattern of “block” since this panel has a “physical” border which cannot allow those moveable repeated patterns to come across the border.



```
Program Status: Drawing Graphic Step by Step
Game Status: Inactive
FPS:20
Pattern: Glider1
Gerneration Number: 34
Live Cell Number: 5
```

```
Program Status: Drawing Graphic Step by Step
Game Status: Inactive
FPS:20
Pattern: Glider1
Gerneration Number: 37
Live Cell Number: 4
```

Choose Your Initial Pattern: Blip, Blip2, Block, Beehive, Loaf, Blinker, [✓ Glider1], Glider2, Glider3, Random, New Pattern

Fig. 3 Glider Pattern will Become Block Pattern at the Boulder

2) The size of the grid in the grid panel is not the same as the size of the grid in the generator, which means the pattern created by it may have some boulder effecting display on the larger panel.

5. References

1. https://en.wikipedia.org/wiki/Genetic_algorithm

2. https://github.com/jsvazic/GAHelloWorld