GA for Life

INFO6205 - Team202

Jianru Wang 001447006
Jing Ren 001447030
Qiuchi Chen 001448400

Introduction

First of all, the GA for life project obey the rule of genetic algorithm and the principles of the game of life.

Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are select

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

Five phases are considered in a genetic algorithm: 1. Initial population. 2. Fitness function. 3. Selection. 4. Crossover. 5. Mutation(We just used mutation in this project)

Game of Life

The Game of Life is kind of artificial game played on a two-dimensional rectangular grid of cells. Each cell has two status — alive or dead. The status of each cell changes each turn of the game (also called a generation) depending on the statuses of that cell's 8 neighbors.

The first generation is the initial pattern(produce randomly in the project) . The second generation evolves from applying the rules simultaneously to every cell on the game board. Afterwards, we obey the rules to create future generations iteratively. For each generation of the game, a cell's status in the next generation is determined by a set of rules. These simple rules are as follows:

If the cell is alive, then it stays alive if it has either 2 or 3 live neighbors
If the cell is dead, then it springs to life only in the case that it has 3 live neighbors

GA for Life Project

Project initializes the first generation of pattern randomly. Then used genetic algorithm to choose the best half of the patterns and mutate their genes to fill another half, these two step is repeated over and over again, util we find a pattern which could exist forever.

We produced Chromosome.java, GA4Game.java, GAScore.java, GAUtil.java, GeneticAlgorithm.java for Genetic Algorithm implementation. And we also produced GameOfLife.java, GOL.java for game of life functions and UI.

Glossary

Gene/Genotype: The binary representation of a phenotype.

Phenotype: a coordinate on a two-dimensional matrix

Pattern: a series of coordinates.

Chromosome: a series of binary numbers represents Pattern

Growth rate: The ratio of number of points exist on the matrix after several generations to the number of points in the first generation.

Reason: represents the outcome of the point survival.

Generation: the number of times that genetic algorithm iterates to produce new pattern.

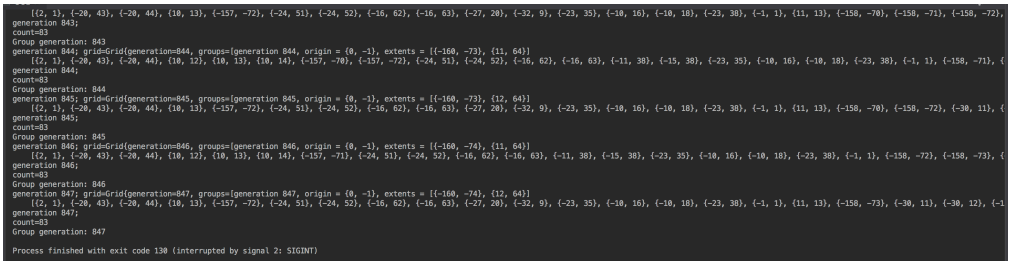
Mutation: the process that change the genotype randomly.

Fitness: the total generation which is iterated before the pattern dies.

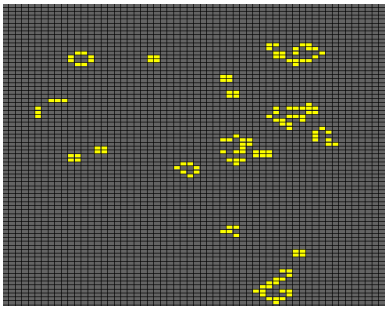
Evolve: the process to choose the best half of patterns and mutation them to fill another half of population.

Project Run

We run project through GOL.java, is the entry of the project. After we run it, first we will see the output at the terminal.



Project is choosing the right pattern which can exist forever and produce more points. After about twenty seconds, the pattern we need is produced and UI shows as below.



The pattern shows on the UI will exist forever and obey the rules of game of life.

Impelementation

Genetic Algorithm:

1. Chromosome.java

The purpose of Chromosome.java is to generate Genotype(a boolean array), to clone the the optimal chromosome you need, to mutate the Genotype randomly, and to get the pattern needed in Game Of Life(Phenotype) from Genotype.

1) Generate Genotype:

The Genotype is a boolean array in this class named “gene”, and in order to get 50 points in the first pattern, the initial length of this gene array is 400.
Also overriding the ‘toString’ function to change the boolean array into a int array which only contains 0 and 1.

```
private void initGeneSize(int size) {
    if (size <= 0) {
        return;
    }
    gene = new boolean[size];
}
```

```
public Chromosome(int size) {
    if (size <= 0) {
        return;
    }
    initGeneSize(size);
    for (int i = 0; i < size; i++) {
        gene[i] = Math.random() >= 0.5;
    }
}
```

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    for(boolean b : gene) {
        sb.append(b ? "1" : "0");
    }
    return sb.toString();
}
```

2） Clone Genotype:

After using ‘fitness’ function(in ‘evolve’ function of GeneticAlgorithm.java) to order all of the genotypes, choosing the better half to be the next generation, and then clone them to use the copy ones to do the next ‘mutation’ step.

```
public static Chromosome clone(final Chromosome c) {
    if (c == null || c.gene == null) {
        return null;
    }
    Chromosome copy = new Chromosome();
    copy.initGeneSize(c.gene.length);
    for (int i = 0; i < c.gene.length; i++) {
        copy.gene[i] = c.gene[i];
    }
    return copy;
}
```

3) Mutation:

After using ‘fitness’ function(in ‘evolve’ function of GeneticAlgorithm.java) to order all of the genotypes and choosing the better half to be the next generation and cloning such better genotypes, according to the number of mutation point you hope, finding the mutation positions randomly and then opposing the values, like ‘true’ turns to ‘false’, and ‘false’ turn to ‘true’.

*: Giving a seed to Random method in GOL class in order to be able to make the behavior of run is repeatable.

```
public void mutation(int num) {
    //允许变异
    int size = gene.length;
    for (int i = 0; i < num; i++) {
        //寻找变异位置
        int at = ((int) (Math.random() * size)) % size;
        //变异后的值
        boolean bool = !gene[at];
        gene[at] = bool;
    }
}
```

4) Get the pattern needed in Game Of Life:

Turn the genotype('gene' array) to the phenotype(used in Game of Life.java)

```java
public String getPattern() {
    int count = 0;
    StringBuilder sb = new StringBuilder();
    for(boolean g : gene) {
        sb.append(g ? 1 : 0);
        count++;
        if (count - 8 == 0) {
            sb.append(",");
            count = 0;
        }
    }
    String bitStr = sb.substring(0, sb.lastIndexOf( str: ","));
    String[] splitBitStr = bitStr.split( regex: ",");
    sb = new StringBuilder();
    for(String str : splitBitStr) {
        String strX = str.substring(0, 4);
        String strY = str.substring(4);
        int x = Integer.parseInt(strX, radix: 2);
        int y = Integer.parseInt(strY, radix: 2);
        sb.append(x + " " + y + ", ");
    }
    String pattern = sb.substring(0, sb.lastIndexOf( str: ","));
    return pattern;
}
```

2. GeneticAlgorithm.java

The meaning of this class is to inherit(the generation numbers cannot be larger than maximum of Iteration times. And then get the most optimal pattern to pass to the Game of Life as Pattern.

1) Calculate(The core function of whole class)

```java
public Chromosome calculate() {
    //初始化种群
    generation = 1;
    init();
    while (generation < maxIterNum) {
        //种群遗传
        evolve();
        print();
        generation++;
    }
    return population.get(0);
}
```

2) Init

In order to complete the above function, at first we need to initialize the whole population.

```java
private void init() {
    population = new ArrayList<>();
    for (int i = 0; i < popSize; i++) {
        Chromosome chro = new Chromosome(geneSize);
        population.add(chro);
    }
    calculateScore();
}
```

3) Evolve(The core is to using fitness to)

3.1 Every time we create a new list to hold the next generation
3.2 Ordering all of the genotypes of this population by fitness. The score of fitness includes ‘reason’, ’generations’, ’growthRate’
The ordering standard is :

If Reason is larger than 2;
Which Generations is bigger;
Which GrowthRate is better;

3.3 Choosing the better half(the half of higher fitness score) as next generations
3.4 Clone the better half in order to make mutation.
3.5 The better half of last generation and the new mutations of them consist the next generation

```
private void evolve() {
    List<Chromosome> nextPopulation = new ArrayList<>();

    Collections.sort(population, (o1, o2) -> {
        if(o1.getScore().getReason() != 2 && o2.getScore().getReason() != 2) {
            return 0;
        } else if(o1.getScore().getReason() == 2 && o2.getScore().getReason() != 2) {
            return -1;
        } else if(o1.getScore().getReason() != 2 && o2.getScore().getReason() == 2) {
            return 1;
        } else {
            if(o1.getScore().getGenerations() == o2.getScore().getGenerations()) {
                if(o1.getScore().getGrowthRate() == o2.getScore().getGrowthRate()) {
                    return 0;
                }
                return o1.getScore().getGrowthRate() > o2.getScore().getGrowthRate() ? -1 : 1;
            } else {
                return o1.getScore().getGenerations() > o2.getScore().getGenerations() ? -1 : 1;
            }
        }
    });

    for(int i = 0; i < population.size() / 2; i++) {
        nextPopulation.add(population.get(i));
    }

    List<Chromosome> temp = new ArrayList<>();
    for(Chromosome chro : nextPopulation) {
        int mutationNum = (int) (Math.random() * maxMutationNum);
        Chromosome clone = Chromosome.clone(chro);
        clone.mutation(mutationNum);
        temp.add(clone);
    }
    nextPopulation.addAll(temp);

    population.clear();
    population.addAll(nextPopulation);

    calculateScore();
}
```

3. GA4Game.java

GA4Game implements the abstract class ‘GeneticAlgorithm’, to get the phenotype from genotype, and to generate behavior(reason, growthRate, generations) bu ‘Run’ function. Of ‘Game’ class.

Setting the range of each point is from (0, 0) to (15, 15), which means from 00000000 to 11111111. The pattern contains 50 points. And because each point contains 8 numbers, so the max length of genotype is 400.

Setting Iteration times as 10 and the number of chromosomes in one population as 4.

```
public GA4Game() { super( geneSize: 400,  popSize: 4,  maxIterNum: 10); }
```

1) changeX

Using ‘getPattern’ function of Chromosome.java class to get phenotype from gentype

```
@Override
public String changeX(Chromosome chro) {
    String pattern = chro.getPattern();
    return pattern;
}
```

2) calculateY

Passing the results of ‘Reason', ‘GrowthRate’, ‘Generations’ of a pattern using fitness function into the GAScore class.

```
@Override
public GAScore calculateY(String x) {

    String patternName = "Genetic Pattern";
    System.out.println("Game of Life with starting pattern: " + patternName);
    final Game.Behavior generations = Game.run( generation: 0L, x);
    GAScore gaScore = new GAScore(generations.getReason(), generations.growth, generations.getGeneration());

    return gaScore;
}
```

4. GAScore.java

GAScore.java is kind of the class which return the score of current pattern.

```
public int getReason() {
    return reason;
}

public double getGrowthRate() {
    return growthRate;
}

public long getGenerations() {
    return generations;
}
```

There are three fields in it, “reason”, “growthRate” and “generation”. Reason describes the cause of the pattern dies, 0 represents all of the points die, 1 represents cells’ activity repeats forever, 2 represents pattern keeps produce new points and unrepeated over 1000 times. GrowthRate describes the changing of the points during game processing.

GrowthRate = points exist / points initialized

Due to the equation above, the bigger growthRate, the better.

Generation represents the iteration times of pattern.

We calculate the three score every time when we invoke calculateY function in the GA4Game.java. CalculateY function will return a GAScore object to save the score for every patterns.

5. GAUtil.java

GAUtil.java use to transform the starting pattern into a two-dimensional array, in order to pass array in the GameOfLife object creation in the GOL.java.

```
public int[][] transform(String pattern) {
    int[][] board = new int[32][32];
    String[] split = pattern.split( regex: ",");
    for(String str : split) {
        String[] s = str.split( regex: " ");
        int x = Integer.parseInt(s[0]);
        int y = Integer.parseInt(s[1]);
        board[x + 8][y + 8] = 1;
    }
    return board;
}
```

Every 8 position in the pattern represents a coordinates for a pint, they are splited by comma, moreover, x coordinate and y coordinate are splited by blank space.

UI

GameOf Life.java

GameOfLife.java includes functions which can be used for game of life and UI button actions. These functions are setCurrentMove, GameOfLife, decide, repain, jPanel1ComponentResized, jButton1ActionPerformed.

GameOfLife:

```
public GameOfLife() {
    initComponents();
    jButton2.setVisible(false);
    offScrImg = createImage(jPanel1.getWidth(), jPanel1.getHeight());
    offScrGraph = offScrImg.getGraphics();
    Timer time = new Timer();
    TimerTask task = () → {
            if(play){
                for(int i = 0; i < hei; i++){
                    for(int j = 0; j < wid; j++){
                        nextMove[i][j] = decide(i,j);
                    }
                }
                for(int i = 0; i < hei; i++){
                    for(int j = 0; j < wid; j++){
                        currentMove[i][j] = nextMove[i][j];
                    }
                }
                repain();
            }
    };
    jButton2.setVisible(false);
    time.scheduleAtFixedRate(task,  delay: 0,  period: 100);
    repain();
}
```

GameOfLife function is used to reference other functions to decide the next moving of every position and implement the game.

Decide:

```
private boolean decide(int i, int j){
    int neighbors = 0;
    if(j > 0){
        if(currentMove[i][j-1]) neighbors++;
        if(i>0) if(currentMove[i-1][j-1]) neighbors++;
        if(i<hei-1) if(currentMove[i+1][j-1]) neighbors++;
    }
    if(j < wid-1){
        if(currentMove[i][j+1]) neighbors++;
        if(i>0) if(currentMove[i-1][j+1]) neighbors++;
        if(i<hei-1) if(currentMove[i+1][j+1]) neighbors++;
    }
    if(i>0) if(currentMove[i-1][j]) neighbors++;
    if(i<hei-1) if(currentMove[i+1][j]) neighbors++;
    if(neighbors == 3) return true;
    if(currentMove[i][j] && neighbors == 2) return true;
    return false;
}
```

Decide function is aimed to calculate the number of neighbors for every point and decide if the point will exist for the next round .

Function checks the left three position of current position first, and then checks the right three position of current position. For the next two step, function checks the bottom position and the upper position of the current position. After these checking, return the number of neighbors.

Repain：

```
private void repain(){
    offScrGraph.setColor(jPanel1.getBackground());
    offScrGraph.fillRect( x: 0,  y: 0, jPanel1.getWidth(), jPanel1.getHeight());
    for(int i = 0 ; i < hei ; i++){
        for(int j = 0 ; j < wid; j++){
            if(currentMove[i][j]){
                offScrGraph.setColor(Color.YELLOW);
                int x = j * jPanel1.getWidth()/wid;
                int y = i * jPanel1.getHeight()/hei;
                offScrGraph.fillRect(x, y,  width: jPanel1.getWidth()/wid,  height: jPanel1.getHeight()/hei);
            }
        }
    }
    offScrGraph.setColor(Color.BLACK);
    for(int i = 1; i < hei;i++){
        int y = i * jPanel1.getHeight()/hei;
        offScrGraph.drawLine( x1: 0, y, jPanel1.getWidth(), y);
    }
    for(int j = 1; j < wid;j++){
        int x = j * jPanel1.getWidth()/wid;
        offScrGraph.drawLine(x,  y1: 0, x, jPanel1.getHeight());
    }
    jPanel1.getGraphics().drawImage(offScrImg,  x: 0,  y: 0, jPanel1);
}
```

Repain function is aimed to repaint the jpanel after every round.

JPanel1ComponentResized:

```
private void jPanel1ComponentResized(java.awt.event.ComponentEvent evt) {
    offScrImg = createImage(jPanel1.getWidth(), jPanel1.getHeight());
    offScrGraph = offScrImg.getGraphics();
    repain();
}
```

JPanel1ComponentResized function is aimed to resize jPanel

Run / Implement

GOL.java

GOL.java is aimed to avtivate the UI.

```
public class GOL {

    /**
     * 这是我们所使用的seed
     * reason: 2
     * growthRate: 0.663
     * genotype: 0111001001001100000111111000101000000101101110011000100110011011100010010001110100100000
     * phenotype: 7 2, 4 12, 1 15, 8 10, 0 5, 11 9, 8 9, 9 11, 8 9, 1 13, 2 0, 8 1, 9 14, 3 6, 8 8, 1 3,
     * seed: 1575750427570
     */
    private static final long seed = 1575750427570L;


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        GAUtil test = new GAUtil();

        /**
         * 这是我们生成seed的时候用的代码
         * 暂时注释掉
         */
//        long seed = System.currentTimeMillis();
//        System.out.println(seed);

        GA4Game game = new GA4Game(seed);
        Chromosome genotype = game.calculate();
        String phenotype = genotype.getPattern();
        int[][] board = test.transform(phenotype);
        System.out.println("================================================================================");

        System.out.println("reason: " + genotype.getScore().getReason());
        System.out.println("growthRate: " + genotype.getScore().getGrowthRate());
        System.out.println("genotype: " + genotype);
        System.out.println("phenotype: " + phenotype);
        System.out.println("seed: " + seed);

        GameOfLife life = new GameOfLife();
        life.setCurrentMove(board);
        life.setLocationRelativeTo(null);
        life.show();
    }

}
```

In order to reproduce the same pattern, we pass parameter “seed” in the creation of GA4Game object, then seed will be passed to the initialization of chromosome, at the end, we will produce genotype randomly with the seed. With the same seed, all of our random processes are drawing from the same generator, the behavior of our project should be repeatable.

Our fixed seed value is shown in the picture, “1575750427570L”.

If we need different seed value, just uncomment this line of code, “long seed = System.currentTimeMillis();”.

After producing the seed, we activate the GA4Game, the implement class of Genetic Algorithm, get the genotype for the project, change the genotype to phenotype, transform the shape of phenotype which name is “borad” in order to use them in the UI. At the end, we will activate the UI class by creating object of GameOfLife class and passing “board” in the object.

Unit Test

ChromosomeTest.java

The purpose of ChromosomeTest is to test the ‘getPattern’, ‘clone’ and ‘mutation’ these three functions.

‘getPatternTest’:

```
// testing GetPattern Function
@Test
public void testGetPattern1() {
    Chromosome chromosome = new Chromosome();
    chromosome.setGene(new boolean[] {true, true, true, false, false, true, false, false});
    String pattern = chromosome.getPattern();
    assertEquals( expected: "14 4", pattern);

}
@Test
public void testGetPattern2() {
    Chromosome chromosome = new Chromosome();
    chromosome.setGene(new boolean[] {false, true, false, true, false, false, true, true});
    String pattern = chromosome.getPattern();
    assertEquals( expected: "5 3", pattern);
}
```

‘cloneTest’:

```
// testing clone function
@Test
public void testclone() {
    Chromosome chromosome = new Chromosome();
    chromosome.setGene(new boolean[] {true, true, true, false, false, true, false, false});
    Chromosome chromosome2 = chromosome.clone(chromosome);
    String pattern = chromosome2.getPattern();
    assertEquals( expected: "14 4", pattern);
}
```

‘mutationTest’:

```
@Test
public void testmutation() {
    Chromosome chromosome = new Chromosome();
    chromosome.setGene(new boolean[] {true, true, true, false, false, true, false, false});
    Chromosome chromosome2 = chromosome.clone(chromosome);
    chromosome2.mutation( num: 10);
    assertFalse(chromosome.getPattern().equals(chromosome2.getPattern()));
}
```

GAScoreTest.java

The purpose of GAScoreTest is to test the ‘getReason’, ‘getGrowthRate’ and ‘getGenerations’ these three functions.

‘getReason Test’:

```
@Test
public void testgetReason() {
    GAScore gs = new GAScore( reason: 0, growthRate: 0.3, generations: 80);
    int reason = gs.getReason();
    assertEquals( expected: 0, reason);
}
```

‘getGeneration Test’:

```
@Test
public void testgetGenerations() {
    GAScore gs = new GAScore( reason: 0, growthRate: 0.3, generations: 80);
    long gr = gs.getGenerations();
    assertEquals( expected: 80, gr);
}
```

‘getGrowthRate Test’

```
@Test
public void testgetGrowthRate() {
    GAScore gs = new GAScore( reason: 1, growthRate: 0.7, generations: 80);
    double gR = gs.getGrowthRate();
    assertEquals( expected: 0.7,gR,  delta: 0.01);
}
```

GAUtilTest.java

The purpose of GATestTest is to test the ‘transform’ function.

```
@Test
public void testtransform() {
    Chromosome chromosome = new Chromosome();
    chromosome.setGene(new boolean[] {true, true, true, false, false, true, false, false});
    String pattern = chromosome.getPattern();
    GATest gt = new GATest();
    int[][] result = gt.transform(pattern);
    int[][]testing = new int[32][32];
    testing[23][13] = 1;
    assertFalse(Arrays.equals(testing,result));
}
```

GA4GameTest.java

GA4GameTest.java test functions which implement in the GA4Game.java and GeneticAlgorithm.java due to GA4Game.java is the implementation class of the GeneticAlgorithm.java.

TestCalculate1 and testCalculate2:

```
@Test
public void testCalculate1() {
    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    Chromosome chromosome = g4g.calculate();
    int i = chromosome.getGene().length;
    assertEquals( expected: 400, i);
}

@Test
public void testCalculate2() {
    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    Chromosome chromosome = g4g.calculate();
    int i = chromosome.getGene().length;
    assertEquals( expected: 400, i);
}
```

These two functions are aimed to test calculate function, if the length of the gene of the return chromosome is 400, we consider the function runs well.

ChangeX1 and changeX2:

```
@Test
public void changeX1() {
    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    Chromosome chromosome = new Chromosome(System.currentTimeMillis());
    chromosome.setGene(new boolean[]{false,true,true,true,true,false,true,false});
    assertEquals( expected: "7 10",g4g.changeX(chromosome));
}

@Test
public void changeX2() {
    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    Chromosome chromosome = new Chromosome(System.currentTimeMillis());
    chromosome.setGene(new boolean[]{true,true,true,true,false,false,false,false});
    assertEquals( expected: "15 0",g4g.changeX(chromosome));
}
```

These two functions are aimed to test changeX function. If the return value equals the target phenotype value, we consider the function runs well.

CalculateY1 and calculateY2:

```java
@Test
public void calculateY1() {

    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    String x = "1 3, 2 4, 3 4, 4 3, 4 2, 3 1, 2 2";
    GAScore score = g4g.calculateY(x);
    assertFalse( condition: score.generations==0);
    assertTrue( condition: score.reason==0||score.reason==1||score.reason==2);
    assertTrue( condition: score.growthRate>=0);

}

@Test
public void calculateY2() {

    GA4Game g4g = new GA4Game(System.currentTimeMillis());
    String x = "1 2, 2 1, 3 1, 4 2, 3 3, 2 3";
    GAScore score = g4g.calculateY(x);
    assertFalse( condition: score.generations==0);
    assertTrue( condition: score.reason==0||score.reason==1||score.reason==2);
    assertTrue( condition: score.growthRate>=-1);

}
```

These two functions are aimed to test calculateY function. If the fields in the return GAScore object is not null, we consider the function runs well.

Reference

https://www.youtube.com/watch?v=FWSR_7kZuYg&vl=en

https://www.geeksforgeeks.org/program-for-conways-game-of-life/

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

https://www.geeksforgeeks.org/genetic-algorithms/

https://www.youtube.com/watch?v=1i8muvzZkPw

https://blog.csdn.net/f641385712/article/details/81115164

https://www.analyticsvidhya.com/blog/2017/07/introduction-to-genetic-algorithm/

https://codereview.stackexchange.com/questions/139125/game-of-life-basic-java-gui-implementation