Project Report
GA Algorithm - Picture Generation

Team Member: Beiwen Guo(001819094)
Zhenyu Zhu(001813361)

- ## Problem Abstract

In our project, we are trying to resolve the picture generation problem using Genetic Algorithm. The goal of our project is to get a target image represented as a collection of overlapping triangles of various colors, transparencies, and locations of vertices.

- ## Implementation

1. **Iinitial State**

   Firstly, we start with 100 random triangles, each of which has 3 random vertices limited by the size of our canvas(255 pixels * 255 pixels) and random colors and transparencies. Then We put all these 100 random triangles into one canvas as our first generation.

2. **Genotype**

   In our project, the genotype is implemented as a combination of the chromosomes of 3 vertices and color(RGBA) of one triangle(candidate). For 3 vertices chromosomes, each of them has 2 individual genes used to respectively represent the horizontal and vertical location of one vertex. For a color chromosome, it is made up of the exact values of red, green, blue and alpha(transparency) refer to and each of values is an individual gene. The below is the partial implementation of the genotype that can indicate what components it includes.

```
public class Genotype {
    private CoordinateChromo one;
    private CoordinateChromo two;
    private CoordinateChromo three;
    private ColorChromo colorChromo;
```

   CoordinateChromo is the chromosome for triangles' vertices:

```
class CoordinateChromo {
    private int vertical;
    private int horizontal;
```

   ColorChromo is the chromosome for triangles' color(RGBA):

```
class ColorChromo {
    private int red;
    private int green;
    private int blue;
    private int alpha;
```

3. **Phenotype**

   The phenotype in our model refers to how one triangle with some certain genotype looks like. It is implemented as a Triangle class. Every phenotype(triangle) has its own genotype as a field. Through its genotype, the phenotype(triangle) can present corresponding traits and looks. At a time, each triangle is also a candidate of some generation canvas(image), so it has a fitness filed to show how well one candidate can match the target image we need to generate.

```
public class Triangle extends Polygon {

    private Genotype genotype;
    private BufferedImage target;
    private double fitness;
```

4. **Expression of genotype to phenotype**

   We use composition OOP mechanism by having an instance of Genotype as a field of Phenotype class(Triangle class) to build the mapping relationship between genotype and phenotype.

5. **Generation**

   We created one Canvas class to represent each generation solution. Each canvas that includes 100 triangle candidates makes up one piece of the generated image. Every time we generate the next canvas, we compare its fitness with the current canvas' s. If the next generation of the canvas has higher fitness, update the current canvas into the next canvas, or keep the current generation unchanged.

```
public class Canvas {
    private BufferedImage target;
    private double matchRate;
    private List<Triangle> triangles;
    private BufferedImage bi;
    private final int width_one = 255;
    private final int height_one = 255;
```

6. **Crossover**

   Because we divide our genotype into 4 different chromosomes, our crossover is based on the chromosome level. For example:

   we have 2 genotypes as parents, each of them has 3 different coordinates representing 3 vertices(3 coordinate chromosomes)and 1 RGBA color chromosome:

   parent 1: (23, 47) (25, 140) (201, 189) (45, 235, 145, 103)

   parent 2: (98, 231) (123, 45), (34, 105) (202, 42, 196, 112)

   For every coordinates and color chromosome, we have a random value, 0 or 1, to determine which parent the child should inherit from. If the random value equals 0, we take the chromosome of the parent1 or take the chromosome of parent2 when it equals 1.

If the random value is 0, 1, 1, 0 for 4 chromosomes, the genotype of the child is (23,47) (123, 45) (34, 105) (45, 235, 145, 103).

For each crossover, the offspring we configured is 2. But after one crossover, we will choose two highest-fitness triangles from parents and children(the number of parents and children for one crossover is 4) as candidates of the next generation.

7. **Mutation**

   In our project, the mutation is the main source of the genetic diversity, which means most of the triangles in one canvas(one generation) will do the mutation. But the mutation result is generally detrimental, so we configure the mutate_rate = 0.01, which means there is only 1 percent possibility for each triangle to process the mutation. If one triangle is determined to mutate, its 4 chromosomes will mutate on the basis of itself based on the different extent of mutating. We have set 3 different extents of mutations, respectively max mutation, middle mutation, and minimum mutation. The extent of mutation is decided through determining if one random value is larger than one standard value we set for the different extent of mutation.

8. **Fitness**

   In our experiment there are two fitness were considered. One is the fitness between a single triangle and the corresponding area on the target image. The other one is the fitness between the whole target image and the generated image. The fitness of a single triangle determines how the gene of this triangle is inherited by its offsprings. The total fitness determines if the new generation should be considered as the parent of the next generation. If the mutations and crossovers did not bring enough good changes, this generation will not be kept.

9. **Selection Function:(Priority Queue)**

   Our selection function selects genes which should crossover with each other. The rest will be selected by a mutation function, which determines if one gene should be mutated and what is the extent of its mutation. In our experiment, the crossover can cause tremendous gene change, so we select genes that we do not want to keep to do this. The rest will be selected to be mutated. To maintain the good genes as much as possible and to keep bringing new genes, the mutation only happens at 1% odd.

- ## Configuration Parameters

  1. **Initial Seed:** 100 initial random triangles
  2. **fractionForCrossover:** 0.02 One pair in 100 triangles crossover
  3. **fractionForMutation:** 0.98 Fourty nine pairs of triangles mutate

4. **Fecundity of mating:** 2 Sexual reproduction generate 2 children per pair
5. **Mutate Rate:** 0.01 The possibility of mutation happens
6. **maxMutateRate:** 0.015 The odd mutation uses the widest range
7. **midMutateRate:** 0.13 The odd mutation uses the middle range
8. **minMutateRate:** 0.5 The odd mutation uses the smallest range

- ## Results

During the process, we find a common thing between the genetic algorithm and the biological evolution. The mutation is always harmful, the wholesome gene is maintained by inheriting; the fraction of the mutation part also cannot be too large, too large fraction will cause the loss of gene which is wholesome and full of potential. However, the mutation is still really important for the evolution of both biology and our image, a generation of good mutation can affect much more than multiple generations of mediocre inheritance.

We finally managed to get an image which is pretty similar to our target, however, since we chose triangle as the elementary composition of our image, the details are not very same. We believe if we use polygons having more edges can improve. Also, the picture we processed is not very complicated, when the target image is larger and has more variations, the converge process will take a longer time and has a chance cannot converge.

Evolution process screenshot: