INFO 6205 Final Project: Genetic Algorithms

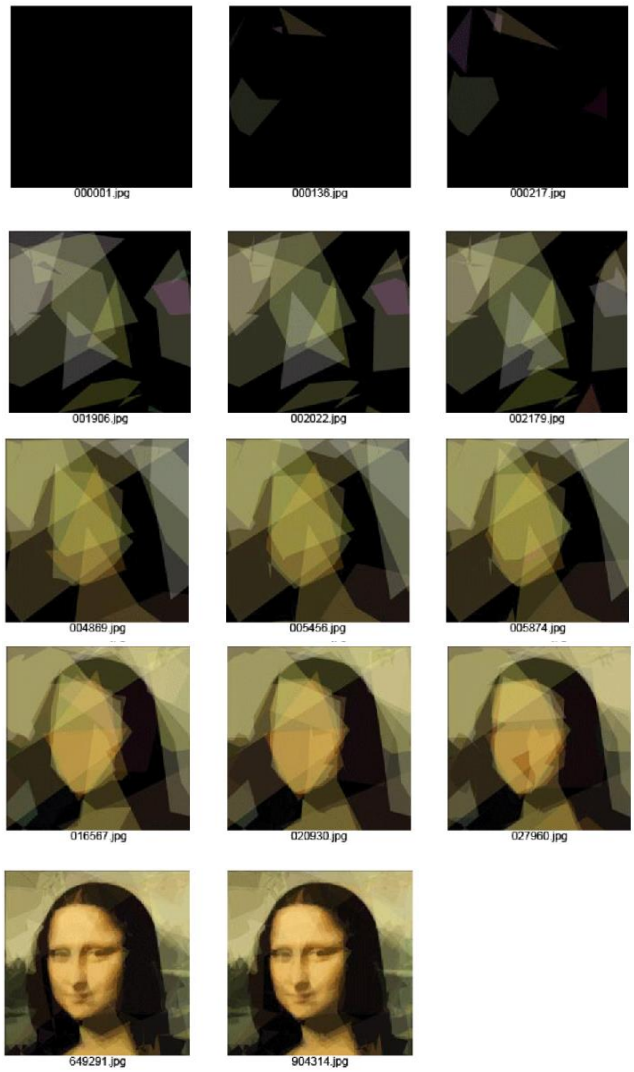
Team 202: Shuhao Xia, Wensong Liu
Github: https://github.com/SummerShoohaw/INFO6205_202

1. Project Proposal

Inspired by Darwin's theory of evolution and Mendelian inheritance, Genetic Algorithms helps to solve problems that are too complicated to address in normal way.

In our project, we aim to use a fixed number of triangle set, in which all triangles are in different shape, color and opacity, to draw a target picture. The idea, however, is also inspired by some genius computer engineer who did this and successfully drew Mona Lisa [1]. However, this idea is so interesting that we decided to implement the same thing by ourselves.

One thing to mention, everything in the project is designed, implemented, tested, debugged, optimized by ourselves, we haven't copied even a single character from anywhere else.



2. Genetic Design

1) Individual: Picture

The individual in our environment is pictures, each picture has its own genotype and phenotype.

2) Gene: Triangle

   In our design, each of the triangle represents for a gene. Each gene has two dimension: location of the triangle, and the color of the triangle.

   Location include 3 pair of coordinates that represents three corners of the triangle, and the range of x value and y value cannot exceed the bound of target picture, also they cannot be smaller than 0.

   We use 32bit ARGB color space to represent colors, so Color has 4 attributes which represents Red, Green, Blue, and Alpha. For Red, Green, and Blue, their values are ranged from 0 -255. For Alpha value, it could also be ranged from 0 – 255, but we give it a more limited range (50 - 150) to make sure the final result will not be affected by any single genes.

3) Genotype: an array of genes (triangle)

   Since we want the number of genes be fixed, so we use array instead of list to store genes. We want to mention that the index of each gene is also very important since the order of triangles will affect the final result.

4) Phenotype: 2d-array of pixels

   The phenotype, the final result in another word, is represented by a 2D array of Color, each cell of the array represents for a pixel, and will finally be drawn on the canvas.

5) Crossover

   The crossover happens between two individuals (pictures), for each of the index, the new born picture will randomly take genes from the parents, means either from parent 1, or parent 2. And the new born will do it in a loop until all of its genes has been assigned.

6) Mutation

   In our system, we have two different kinds of mutation, one is global mutation, another one is called minor mutation. The idea of having two different mutations came very late to our mind, but they were much useful and accelerates the evolution. The reason why we design two kinds of mutation will be elaborated at the end of the report [2].

   a) Global mutation

      During global mutation, every attribute of that specific gene will be assigned to another randomly selected value from its value range. For example:

      - Red will be assigned a random value from 0 – 255, same applies for Blue and Green
      - Alpha will be assigned a random value from 50 – 150.
      - Those 3 pairs of coordinates will also be re-assigned to another three pairs of randomly generated coordinates, and collinear will be detected if happens, in the case of collinear, one of three pairs will re-generate until they can form a triangle.

   b) Minor mutation

      During minor mutation, every attribute of that specific gene will be changed slightly according to its original value. For example:

      - For Red, the program will generate two random integers ranged from 0 – 100, let's say random1 and random2, and newRed = oldRed + random1 – random2. Same applies for Blue and Green.

        Through this approach, there are some possibility that it might change a lot, but

in most case, the value will change around its original value. Outliers will be detected, any new value that is below than 0 or bigger than 255 will be assigned to 0 or 255.

- Same approach applies to Alpha, but random1 and random 2 are ranged from 0 – 25. Outliers will be detected as well.
- Considering 3 pairs of coordinates, for each of the coordinate, its value will multiply a factor which is randomly ranged from 0.8 to 1.2 to generate the new value. More precisely speaking, it multiply 80-120 firstly and then divided by 100 to make sure it is still an integer.

7) Nature

Our nature, or environment, has several important configuration attributes to make sure the evolution and elimination goes smoothly, rapidly and correctly.

- Generation number: this number is very important, we keep track of the number of current generation in the system, this is a very good sign to show whether our algorithm is running faster than before or slower than before, ideally, we use it as a factor to evaluate our application.
- Target individual: the target picture will be read and saved as an attribute in the environment.
- Max population: since there will be hundreds of thousands evolutions and eliminations, a fix number of the maximum population is important to make sure the system will not crash or run out of memory and also a good guarantee for the speed.
- Terminate generation: this attribute is rather useless but we still keep it in our system, we want to give users (who has some interests in trying our application) a chance to set up the terminate point. Actually most time the app will be terminated by ourselves.
- Mutation factors: this is another very very very important attribute in the system, and also the magic power to make sure the evolution goes smoothly, rapidly and correctly. The factor will change in relation to the fitness of current generation. its magic power will be elaborated at the end of the report [3].

8) Fitness (natural selection)

Fitness, or evaluation of individual, is the most struggling part for genetic algorithms, or at least for our project. We have tried several approaches to evaluate individuals, we believe there are better ways to do that, but currently we are going to simply share our design (at the end of the report [4]) :

- By sum difference.
- By average ARGB distance.
- By giving bonus scores in relation to weighted ARGB distance.
- By HSV distance.
- By weighted HSV distance.
- By reciprocal of ARGB distance.

3. Implementation

1) Framework:

We have build the basic frame of Genetic Algorithms, we have 6 abstract classes including Gene, GenoType, PhenoType, Nature, Individual and Nature. All mandatory functions are

declared in the abstract class, so the frame can be re-used for other GA topics.

Please check package src/GAFrame for reference of our framework. And also src/GASample for our implementation of GAFrame

2) The genetic code and a random generator/mutator of such code:

We have build a GAHelper class, the helper would take the dimension of the target picture, as well as the number of genes we want in an individual. After that, we can use this helper class to configure our original environment. In the configure function, the initial population size and the target picture is passed, the certain amount of individual will be generated and target picture will be set up also.

Please check source code in src/GateWay/GAHelper.java

3) Gene expression:

I want to re-explain to whom is reading the report of what is our geno and what is our pheno:

- The genotype, or list of genes, is simply a bunch of triangles, each of them has its own color and location, they can be overlapped, they can be coincided, there is no limit, but PLEASE UNDERSTAND THAT THEY HAVE ORDER, and the order will never be messed up, the index of a specific gene will always be the same in any individuals in future generations.
- The phenotype, or the metrix of pixels, is simple a bunch of colors, each element of the matrix represent for a specific pixel of the picture, and the value stored inside is the color of that pixel. These values have nothing related with any single triangles but the total reflection of all triangles.

So here is how we define gene expression: the phenotype is initialized with all colors are white (think of it as a white canvas), and then, from gene index 0 to gene index last, each gene is 'placed' one by one on the 'canvas', the final color will be calculated as per ARGB color mixing rule [5].

4) The fitness function:

Please refer to appendix [4].

5) The survival function:

We have a elimination function in our environment, which is pre-defined as a abstract function in the father class Nature.java.

In our case, we simple delete the lowest 1/3 individuals as per their score (fitness)

6) The evolution driver:

The evolution takes place immediately after elimination, it will never stop (in general).

7) A logging function to keep track of the progress:

We have a function called toFile() under Picture.java to draw any individual to a .png file, so we can easily view it and evaluate it by human eye, which is far more clear than reading pixel values.

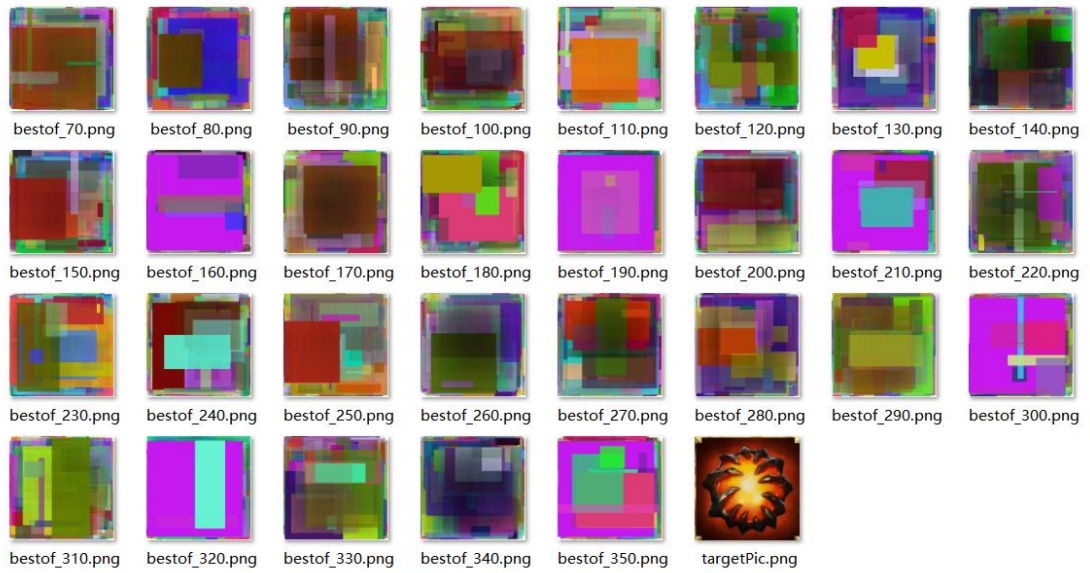
8) Unit tests:

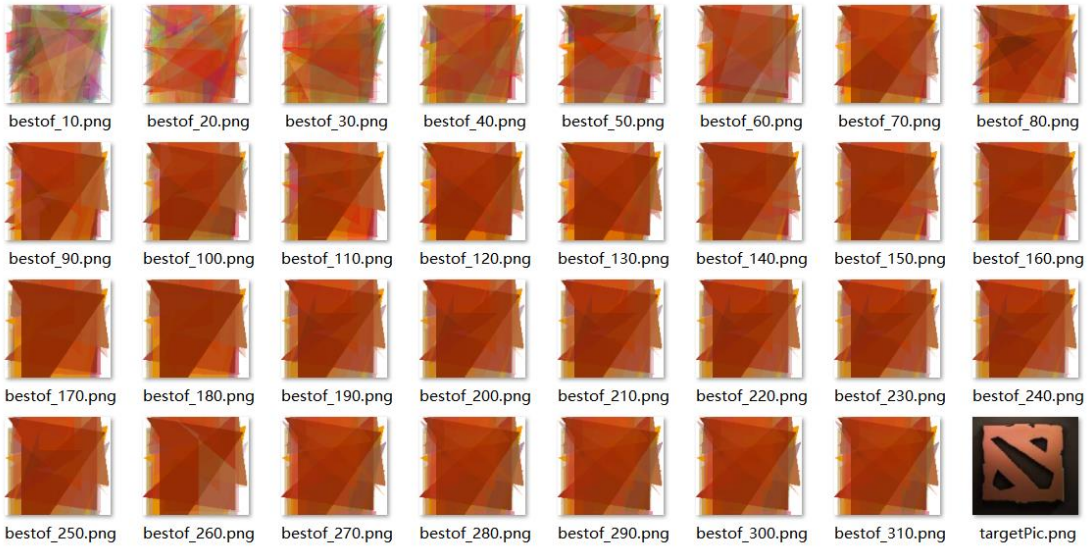
There aren't so many unit tests in our project since we write code in a very cautious way, but any how we have 4 unit tests to test expression, cross, mutation, drawing, and resize.

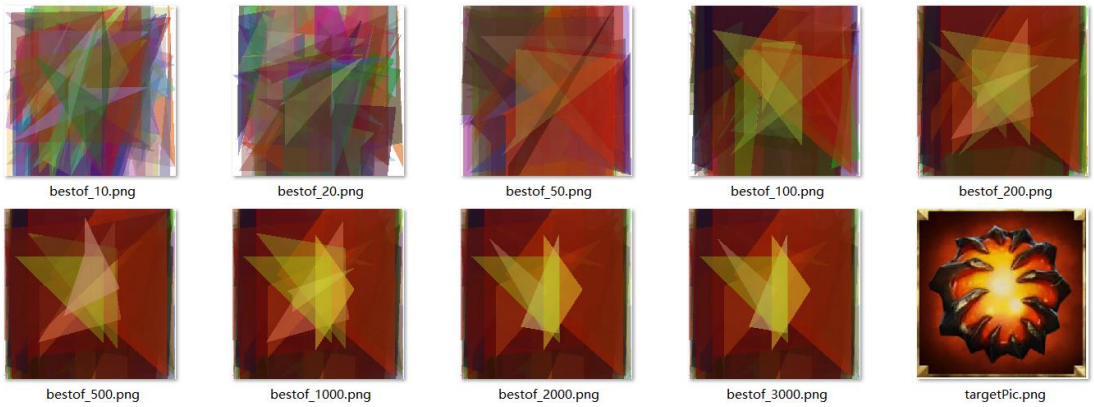
4. Results

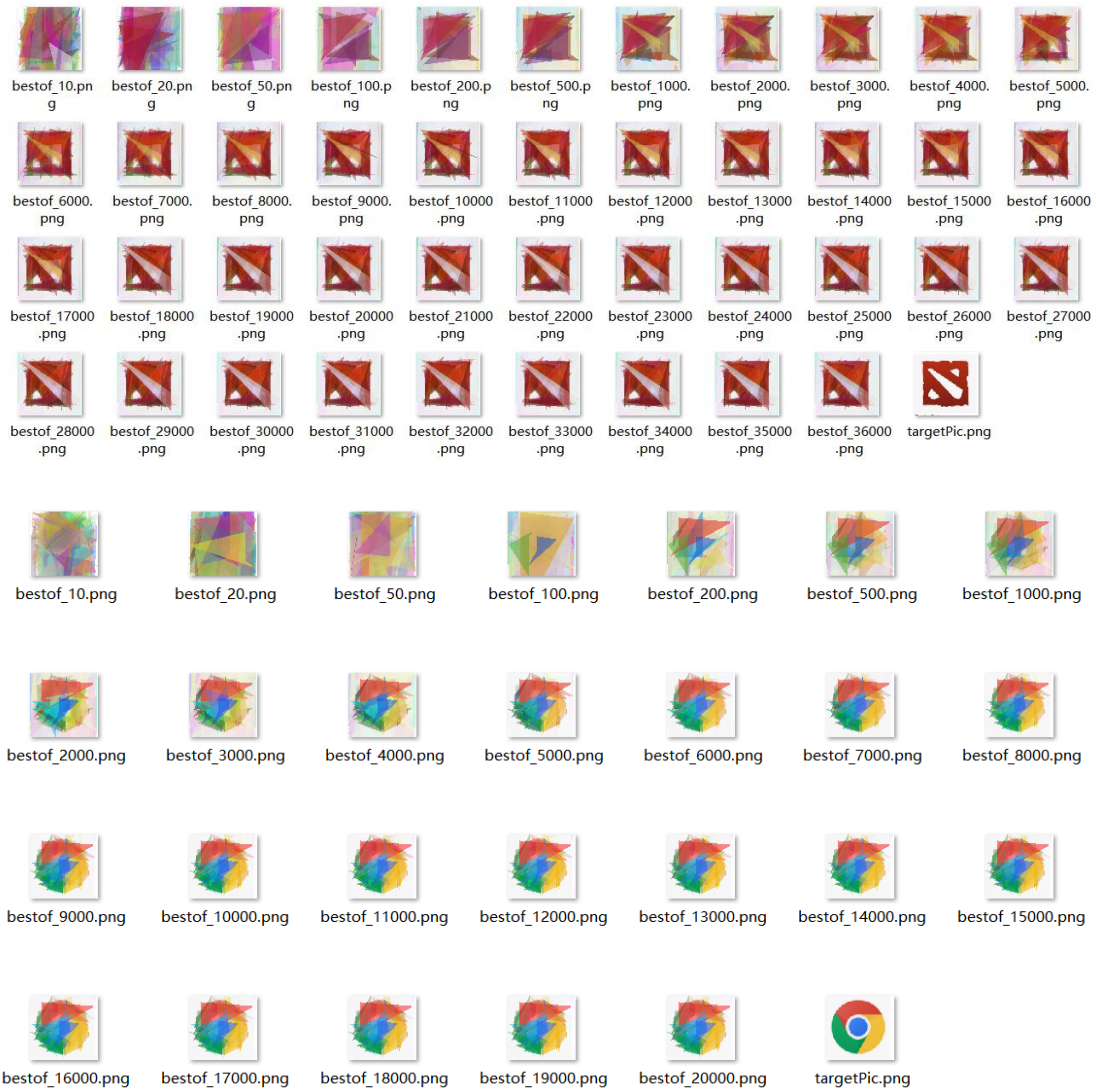
We ran our application against several target pictures, we will show some failed examples, and followed by some good ones, please see below results (filename of

the picture represents the generation) :









5. Appendix

[1] For Mona Lisa simulation, please refer to this website for more details: https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/

[2] Reason to design two kinds of mutation:

In the beginning of evolution, every gene is randomly generated by our system, so they might be far from what we want (target image), both its location and color, so in the beginning, we want the mutation to be a global mutation in order to increase the chance that certain kinds of genes can be shown up during evolution.

However, when the evolution goes to several thousands, the shape slowly slowly becomes to what we exactly want but only some detail of the picture needs changing, this time we don't want global mutation since it mostly will give us a worse gene but not a better one. We would just do a minor mutation, the gene will do some slightly change around its original values.

[3] Mutation factors:

The factors are mutation possibility and number of gene that's going to mutate if mutation happens to an individual.

1) Possibility: in the beginning of evolution, we set the possibility rather high, because we want to amplify our gene library, we eagerly expect the mutation can happen in most of the time. However, as the picture comes closer and closer to what we want, wo reduce the possibility (according to mid score of the whole population) step by step to 10%, we still want to keep a 10% chance in the end, because we need mutation to help us get better genes.
2) Number of genes going to mutate when mutation happens to an individual: Each individual has a fix number of genes (triangles), in our experiment, we set it to 100. In the early phase of evolution, the factor is set to 16, we will step by step reduce it to 1. This is mandatory, if the number is high, many genes are going to mutate, so even if we luckily get one gene mutated just as what we want, but the individual might be eliminated because other 15 genes are mutated to some very very bad condition. We want our good gene to be inherited to future generations.

[4] Fitness calculation:

R – Red, G - Green, B - Blue , A - Alpha

a) By sum difference.

$$\text{fitness} = \sum[(R_i - R_t) + (G_i - G_t) + (B_i - B_t) + (A_i - A_t)]$$

in this case, the higher the worse.

b) By average ARGB distance.

$$\text{fitness} = \frac{\sum[(R_i - R_t)^2 + (G_i - G_t)^2 + (B_i - B_t)^2 + (A_i - A_t)^2]^{1/2}}{total\ number\ of\ pixels\ \times 4}$$

in this case, the higher the worse.

c) By giving bonus scores in relation to weighted ARGB distance.

In this case, calculate the different in each dimension for each pixel, Diff-alpha means the distance of two alpha values, and Diff-color means the Euclidean distance of R, G and B. After when:

Diff-alpha < 100, fitness increments by 1
Diff-alpha < 20, fitness increments by 2
Diff-color < 20, fitness increments by 1
Diff-color < 10, fitness increments by 2
Diff-color < 5, fitness increments by 3

After looping all the pixels, we can get a final fitness, which represents the score.

In this case, the higher, the better

d) By HSV distance.

After using ARGB to calculate fitness, we started thinking about using other color models, such as HSV, we transformed ARGB color space to HSV color space, and calculated the Euclidean distance.

In this case, the higher, the worse.

e) By weighted HSV distance.

Same as d), we calculate the Euclidean distance similarly but we give H, S, V different weight, H has a weight of 3, S and V have a weight of 0.5.

In this case, the higher, the worse.

f) By reciprocal of ARGB distance.

Similarly to what we have in a), we calculate the reciprocal of the value, which is:

$$\text{fitness} = \frac{total\ number\ of\ pixels\ \times 4}{\sum[(R_i - R_t)^2 + (G_i - G_t)^2 + (B_i - B_t)^2 + (A_i - A_t)^2]^{1/2}}$$

And actually this one is what we currently have in the system. **In this case, the higher, the better.**