Introduction to AI
Assignment #2 report
By Danil Ginzburg, BS17-07

How I see art…

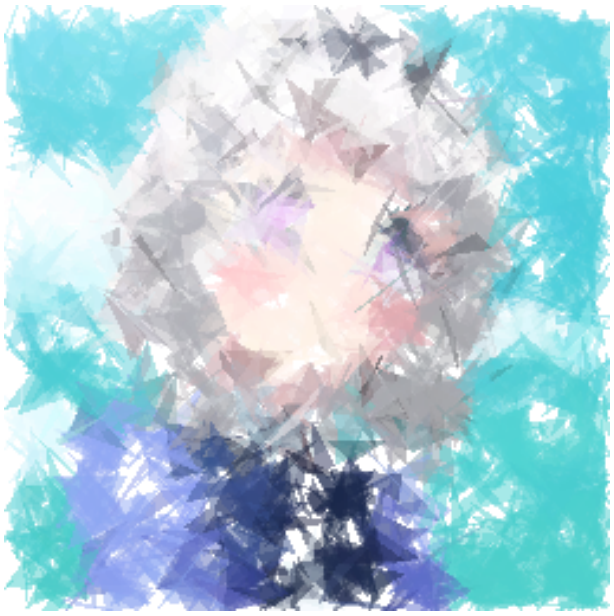
For me, art is something that makes me think or imagine. Most often, art is open for interpretations. In other words, the True Art does not always have to be specific, the only thing it should do - is to convey feelings.

My idea for this task, is to decrease specificity of images as much as possible in some creative way, but in the same time leave the same paint. In other words, to make image more abstract. Although I do not pretend for the True Art, but I have tried my best to implement this idea in this assignment.

Results so far

About 350 generations

About 500 generations



About 200 generations

Main tools

In this assignment I used

- *Python 3.6*
- *opencv* library for image reading/writing
- *numpy* library for image processing and other numerical computations
- *copy* library for copying parent's genes to child
- *random* library for generating random numbers
- *tqdm* library for progress bar

How to run

python3 main.py [--path image/path] [--generatons 500]

The result image will appear in the same folder with result.jpg name.

Genetic algorithm in python

Structure of code

I have 3 classes:

- Gene
  - Stores location of a triangle (three points) and color of it. Represents a triangle on image.
  - On creation time of a new gene, creates random location and color
  - Has function «mutate» which changes somehow location and color
  - Has function «draw» which draws itself on some image
- Chromosome
  - Genes set. Represents an art image.
  - Has function «get_image» which overlays all its genes (triangles) on white background
  - Has function «get_fitness» which calculates fitness of a chromosome
  - Has function crossover which returns new chromosome based on parents
  - On creation time generates N genes
- GeneticImage
  - Accumulates all necessary functionality for genetic algorithm
  - On creation time generates 2 chromosomes (initial population)
  - Has function «generate_population» which yields new population on every iteration
  - Has function «best_chromosome» which finds the best chromosome over population

Default global variables

```
#  Definitions
IMAGE_SIZE = (256, 256)
mutation_chance = 0.2
num_genes = 2500
num_generations = 50
gene_deviation = 16
location_mute = (-4, 4)  # define borders for location mutation
alpha = 0.5
```

IMAGE_SIZE is working size for images, later on program will resize result image to (512, 512). *gene_deviation* is for how much triangle's points can deviate from its center (like radius of triangle). *location_mute* says how much a point of a triangle can mutate at a time. *alpha* defines alpha channel. The rest variables do not need my explanation.

Fitness

Just after creation of a new chromosome fitness is calculated:

```
class Chromosome:

    def __init__(self, genes=None):
        if genes is None:
            self.genes = [Gene() for _ in range(num_genes)]
        else:
            self.genes = genes
        self.fitness = self.get_fitness()
```

Named argument «genes» is used when creating a child and we need to specify future genes beforehand. By default it is None and then random genes are generated. Then we calculate fitness.

```
def get_fitness(self):  # less fitness -> better
    diff = np.sum((image.astype("float") - self.get_image().astype("float")) ** 2)
    diff /= float(IMAGE_SIZE[0] * IMAGE_SIZE[1])
    return diff
```

Fitness function computes squared difference between original image and image composed of triangles

Crossover

```
def __add__(self, other):  # crossover
    genes = deepcopy(self.genes) \
        if random.choice([0, 1]) else deepcopy(other.genes)
    for gene in genes:
        gene.mutate()
    return Chromosome(genes=genes)
```

Crossover function is implemented as «magic method» in python. So that I can write parent1 + parent2 and it will return the result of this function.

This function takes randomly either mother's genes or father's. Then mutate all genes and return new chromosome.

Mutation

```
def mutate(self):

    y = (self.location[0][0] + self.location[1][0] + self.location[2][0]) // 3
    x = (self.location[0][1] + self.location[1][1] + self.location[2][1]) // 3

    if random.random() < mutation_chance:
        # mutate color

        r, g, b = image[x, y]
        self.color = (int(r), int(g), int(b))

    if random.random() < mutation_chance:
        # mutate location
        ...

    self.location = [tuple(pt1), tuple(pt2), tuple(pt3)]
```

First, I find center of my triangle, x and y will be coordinates of that center.

Secondly, if I am lucky (there is some chance of it defined by «mutation_chance» global variable) I mutate color. I set [r, g, b] to the pixel's [r, g, b] which is in the center of the triangle.
Thirdly, I mutate location. Randomly. But since my triangles are not very big, I keep their size under control like that:

```
elif pt1[0] - y > gene_deviation:
    pt1[0] = y + gene_deviation
```

Selection

```
def generate_population(self):
    for generation in tqdm(range(num_generations), unit='generation'):
        child1 = self.population[0] + self.population[1]
        child2 = self.population[0] + self.population[1]
        child3 = self.population[0] + self.population[1]
        if child3.fitness < child2.fitness and child3.fitness < child1.fitness:
            self.population = [child1, child2]
        elif child1.fitness < child2.fitness and child1.fitness < child3.fitness:
            self.population = [child2, child3]
        elif child2.fitness < child1.fitness and child2.fitness < child3.fitness:
            self.population = [child1, child3]
        yield self.population
```

I always do have exactly 2 chromosomes in population. In every generation I create 3 new chromosomes: child1, child2 and child3. I compare their fitnesses and choose best 2 of them to represent my new population.

Links

Git: https://github.com/thedownhill/ai_genetic_art_picture