Yasincan Bozkurt
2304202

EE 449 Homework 2

Number of Individuals: This is the population size for each generation. A greater population size may inject more variation into the population, which would then cover a bigger portion of the problem space. Larger populations call for more computer power, though.

Number of Genes: This variable defines how complicated each person is. An person can represent more complicated solutions the more genes they have. However, it can also widen the search field and slow down optimization.

Tournament Size: In tournament selection, a subset of the population is randomly picked, and the top candidate is chosen from this subset. A greater tournament size may increase selection pressure (better candidates have a higher chance of being chosen), but it may also reduce population diversity.

Number of Elites: The greatest are immediately passed on to the next generation via elitism. The best solutions won't be lost as a result of crossover and mutation stochastic procedures by increasing the proportion of elites. But excessive elitism can reduce population variety and hasten the convergence of less-than-ideal solutions.

Number of Parents: The number of parents who will produce children is determined by this parameter. If the parents are not well chosen (e.g., based on their fitness), having more parents can potentially result in a deterioration in the general fitness of the population. More parents can introduce greater genetic variety into the following generation.

Mutation Probability: Individuals undergo random changes as a result of mutation, which encourages population variety. By examining more of the solution space, a greater mutation rate can assist in avoiding local optima. A mutation rate that is too high, however, may damage the fundamental elements of sound solutions and prevent the algorithm from convergeing.

Mutation Type: The optimization process can be affected in a variety of ways by various mutation techniques. You're presently utilizing "guided" mutation, which modifies the current values to enable a finer search but perhaps slower convergence. The values are reset randomly through "unguided" mutation, allowing for a wider search but with the potential to stymie promising answers.

Suggestion 1: Adaptive Mutation Rate

Exploring the solution space requires careful consideration of the mutation rate. When the mutation rate is too high or too low, the algorithm may act more randomly and take longer to get the optimal solution. Adaptive mutation rates change according to how well the algorithm performs. For instance, the mutation rate may be temporarily raised if the greatest fitness does not advance after a given number of generations.

Suggestion 2: Fraction of Elites

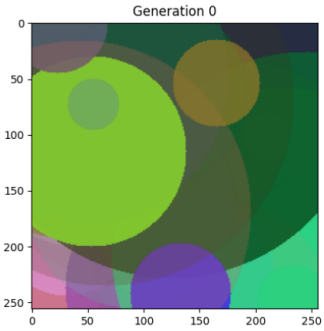
We will be able to modify the strength of selection using the fraction of elites. To get a better outcome, we will alter the proportion of elites (parents) as low (high) as possible from the beginning to create more variety. In other words, rather than entering the tournament as a warrior, we are urging people to enroll early. We will raise (lower) or decrease this proportion as the number of iterations rises. The convergence time will be shorter as a consequence.

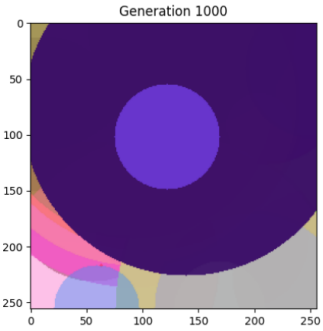
Suggestion 3: Using Different Selection Method

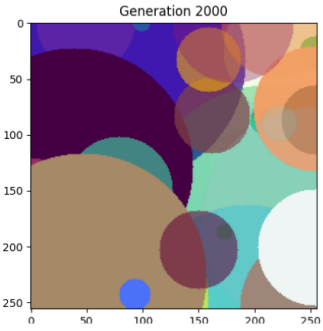
A tournament selection approach is used in the present selection process. To test out other selection strategies, though, could be advantageous. For instance, roulette wheel selection, where the probability of a person being chosen is proportionate to their fitness, may aid in preserving population variety and delaying early convergence.

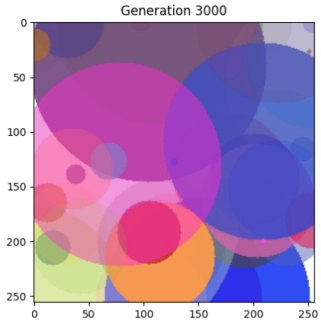
Yasincan Bozkurt
2304202

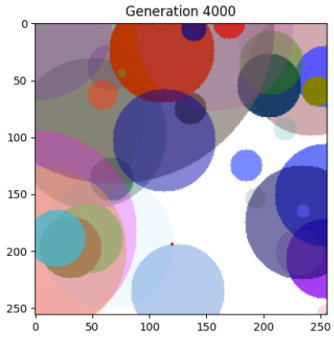
number of individuals =10

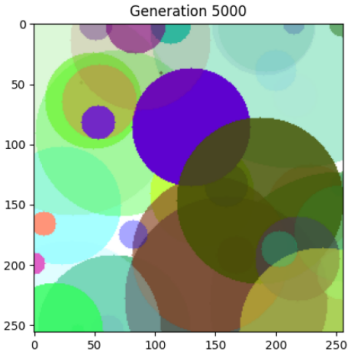


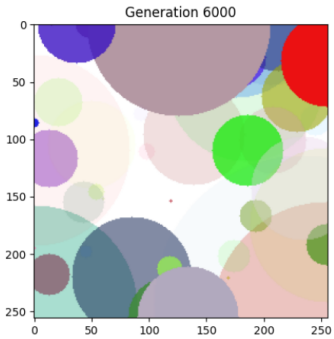


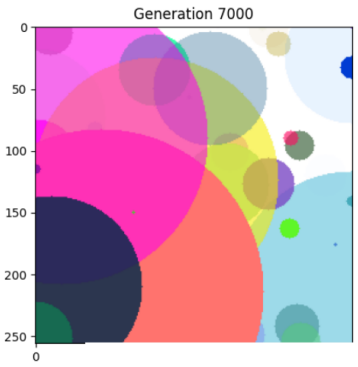


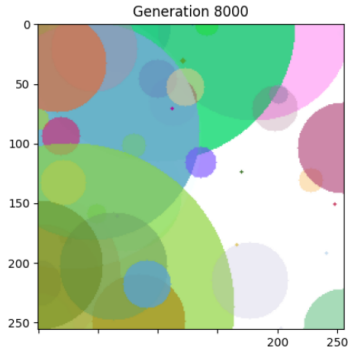


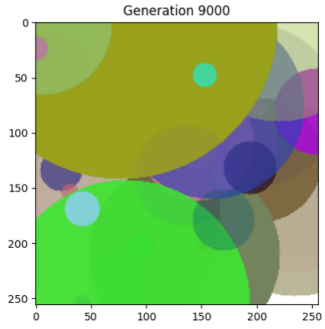


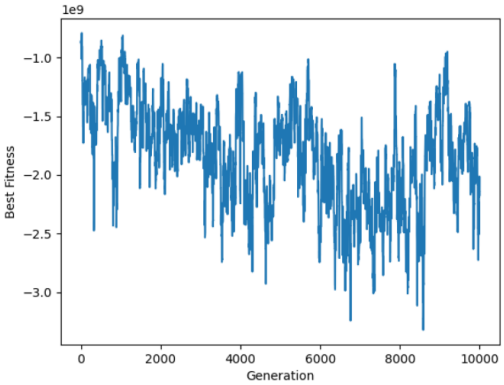






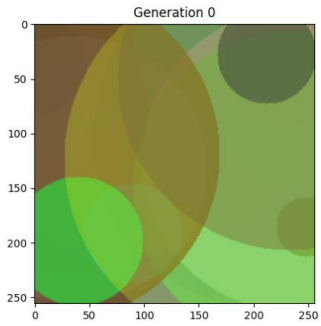


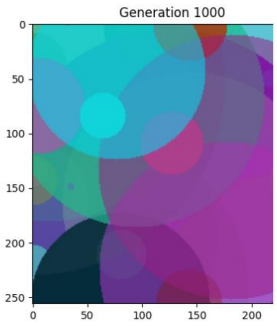


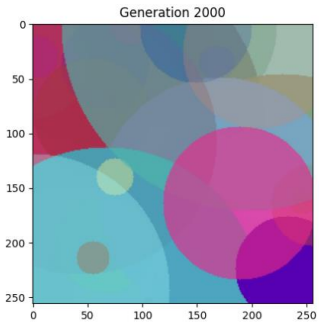


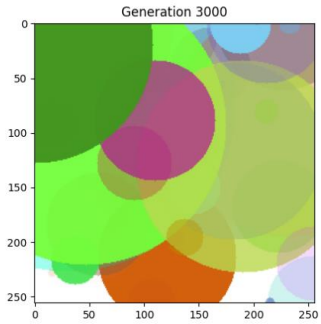
Yasincan Bozkurt
2304202

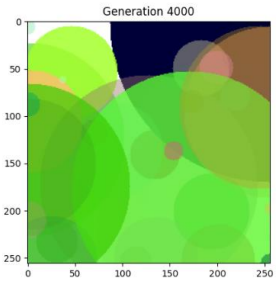
Default Setting

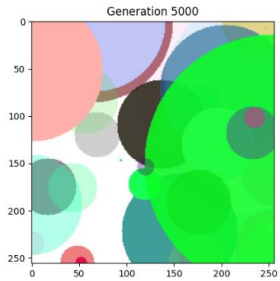


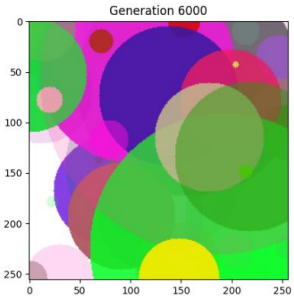


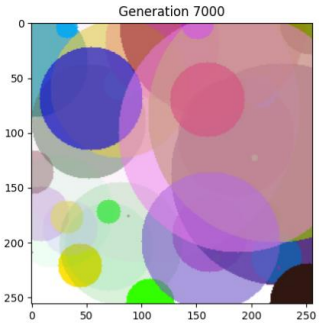


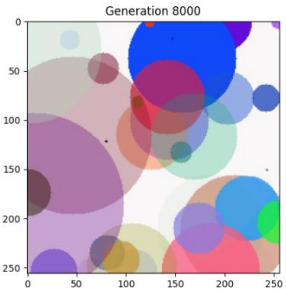


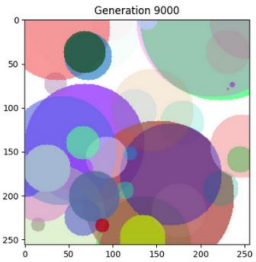


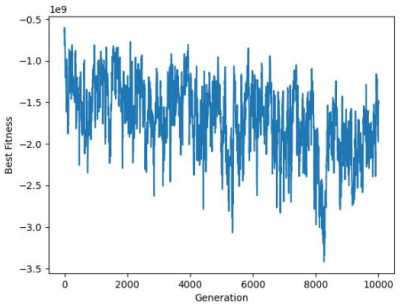






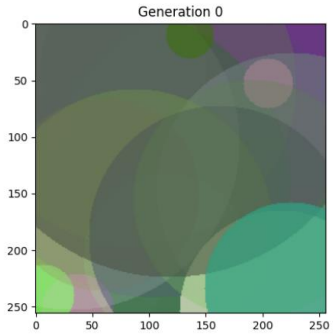


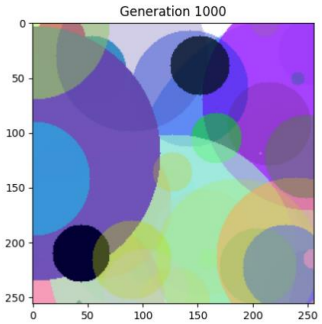


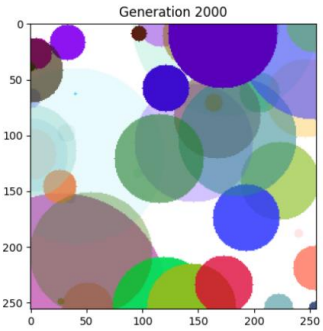


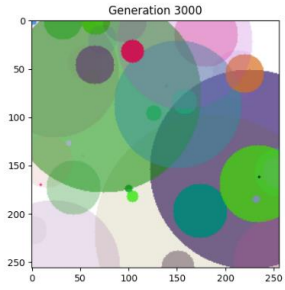
Yasincan Bozkurt
2304202

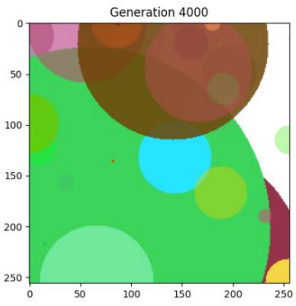
Number of Individuals 40

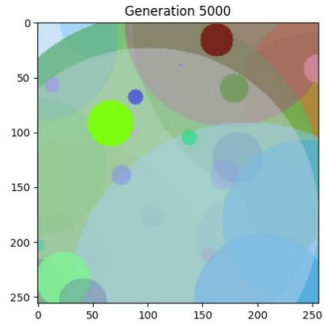


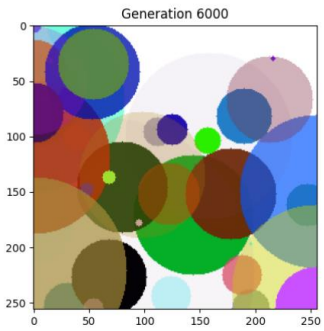


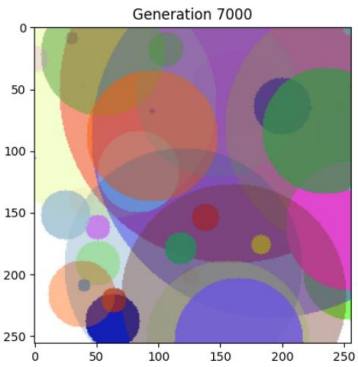


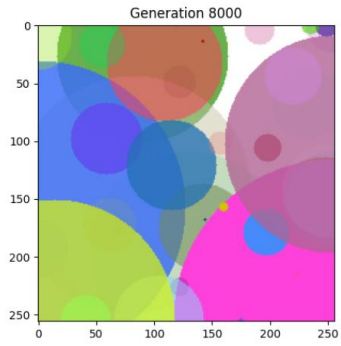


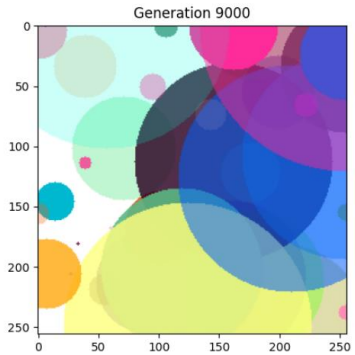


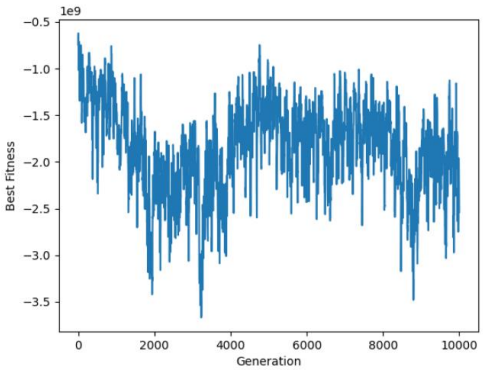






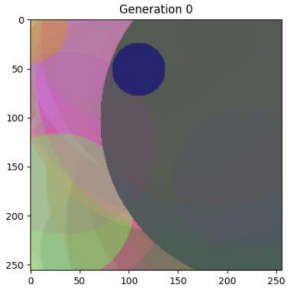


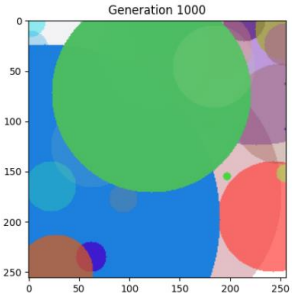


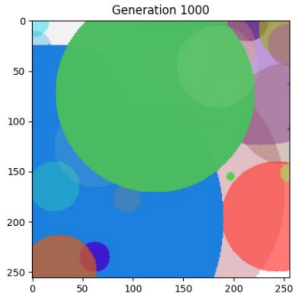


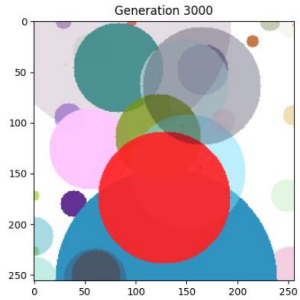
Yasincan Bozkurt
2304202

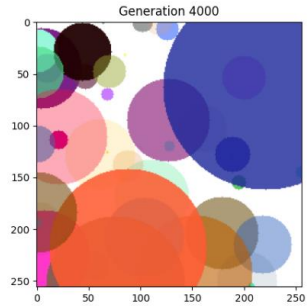
Number of Individuals 60

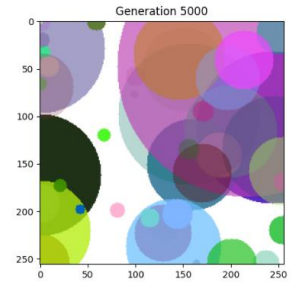


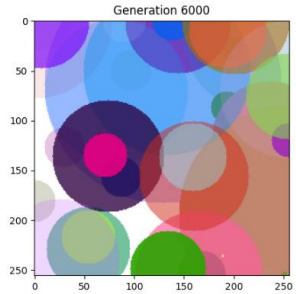


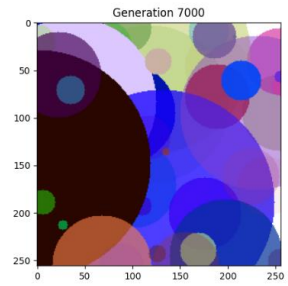


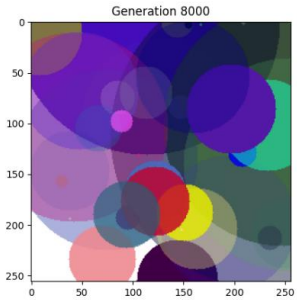


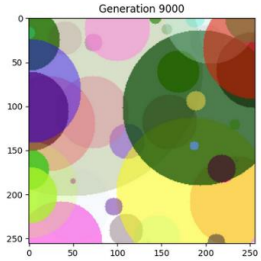


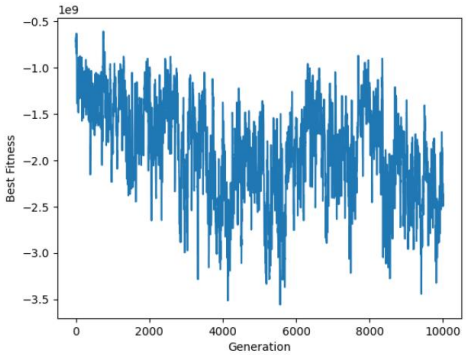






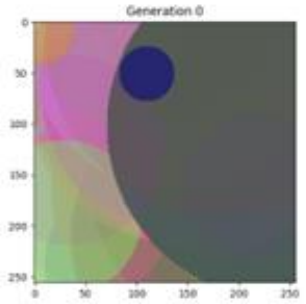


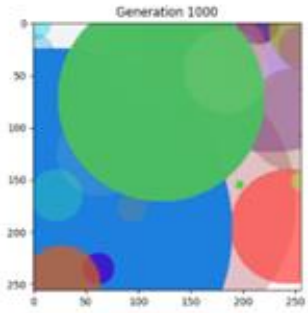


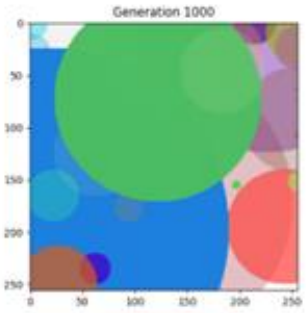


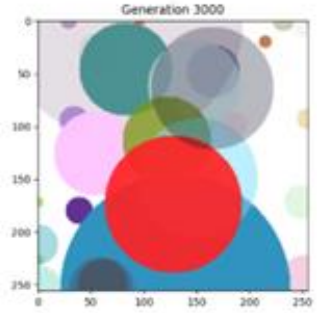
Yasincan Bozkurt
2304202

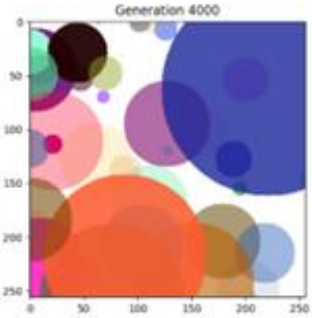
Number of Individuals 80

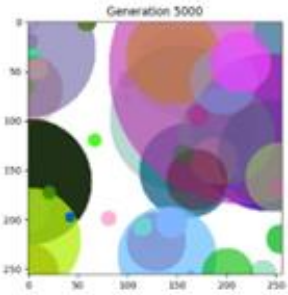


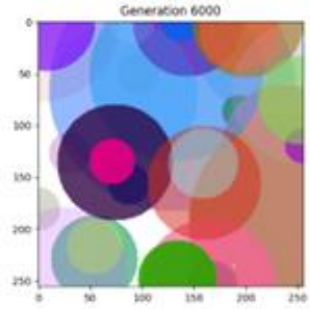


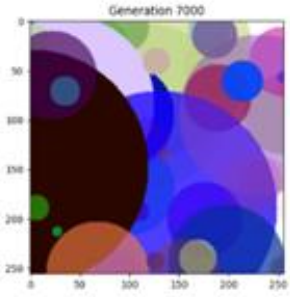


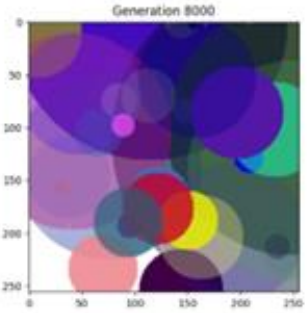


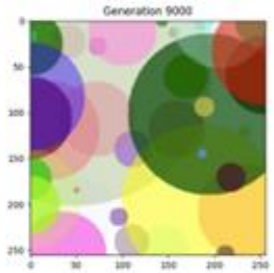


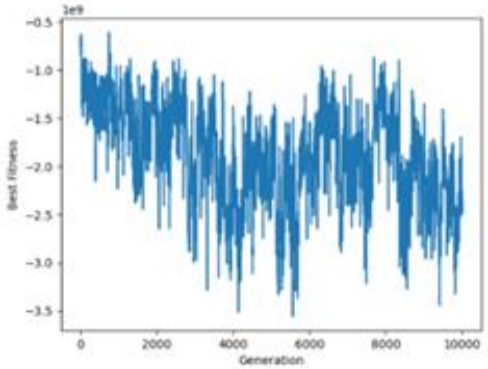






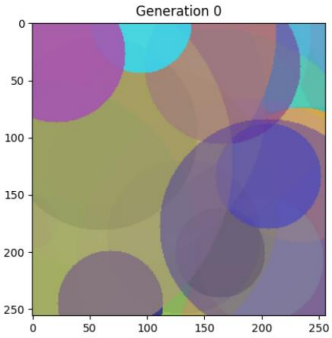


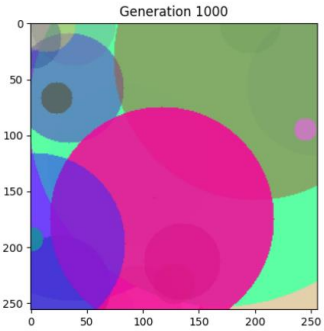


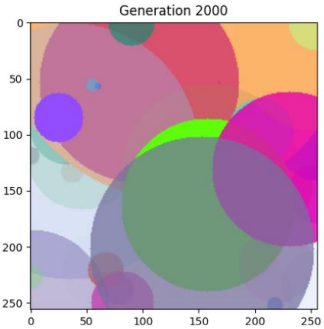


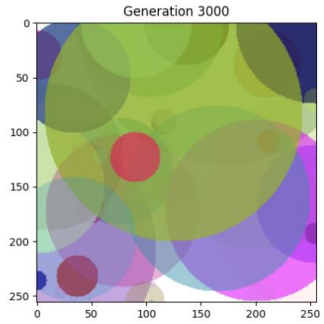
Yasincan Bozkurt
2304202

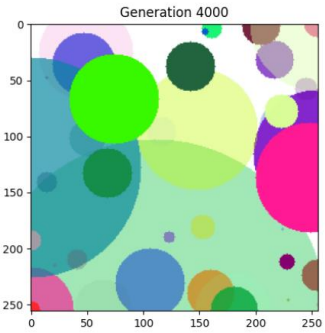
Number of Individuals = 5

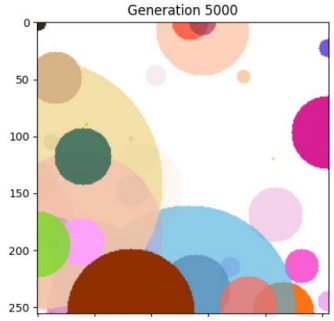


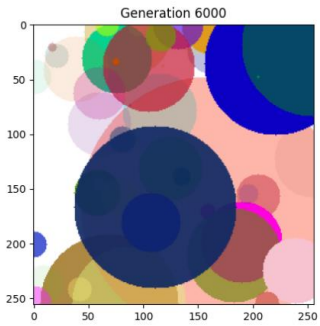


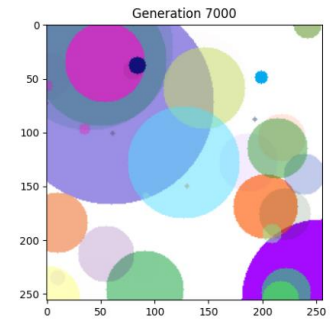


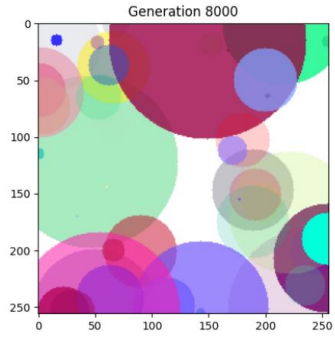


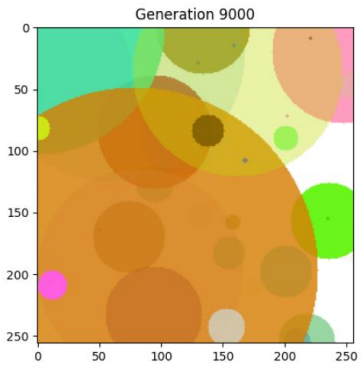


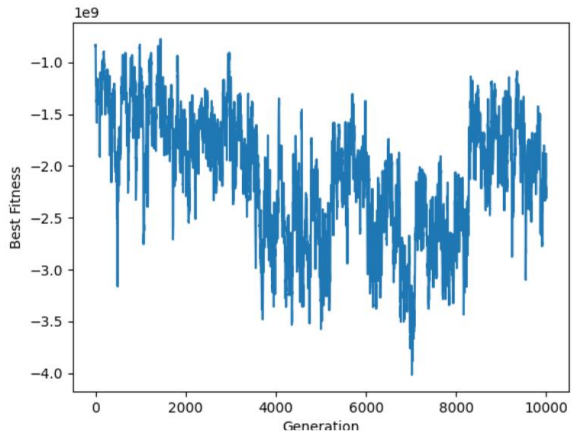






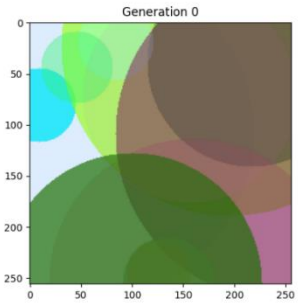


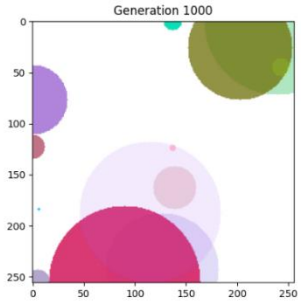


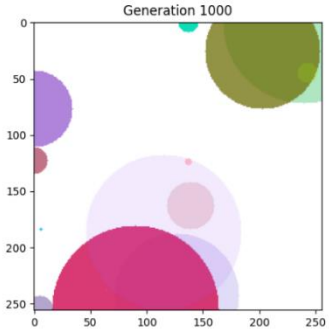


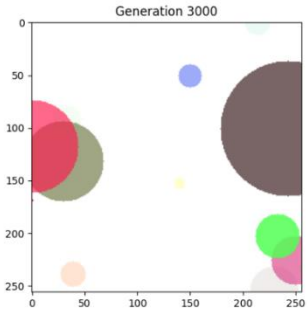
Yasincan Bozkurt
2304202

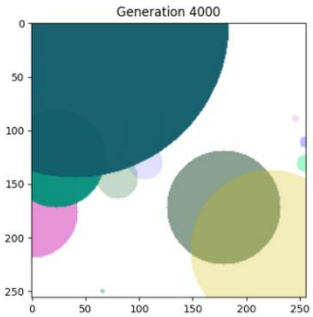
Number of Genes = 15

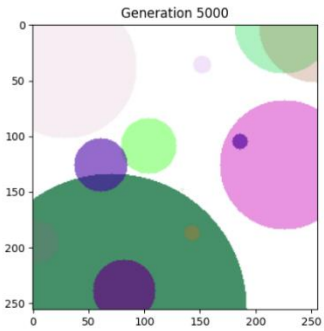


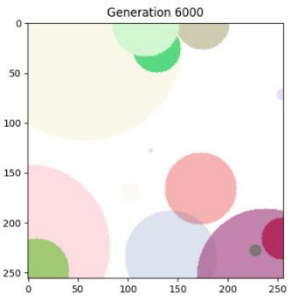


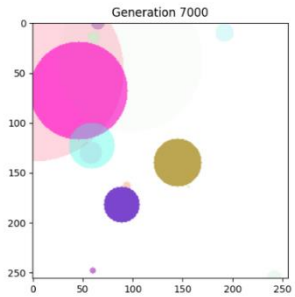


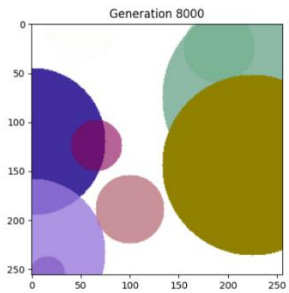


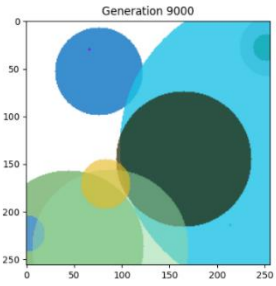


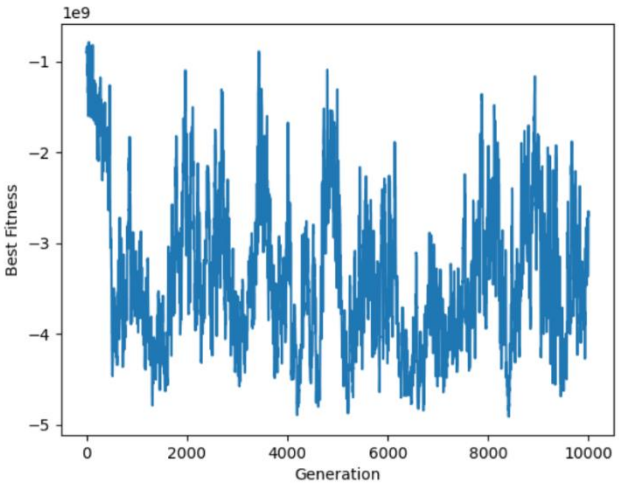






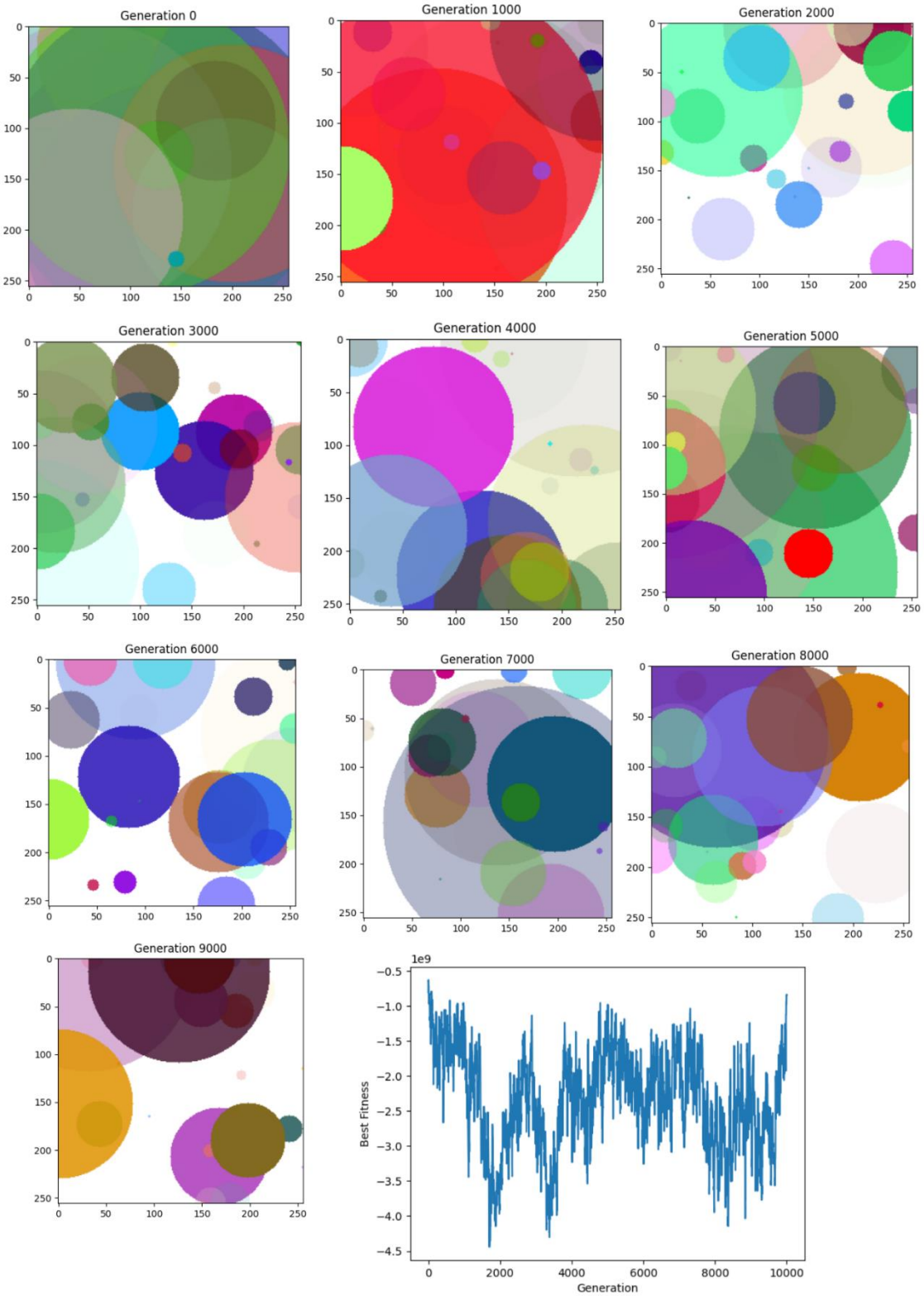






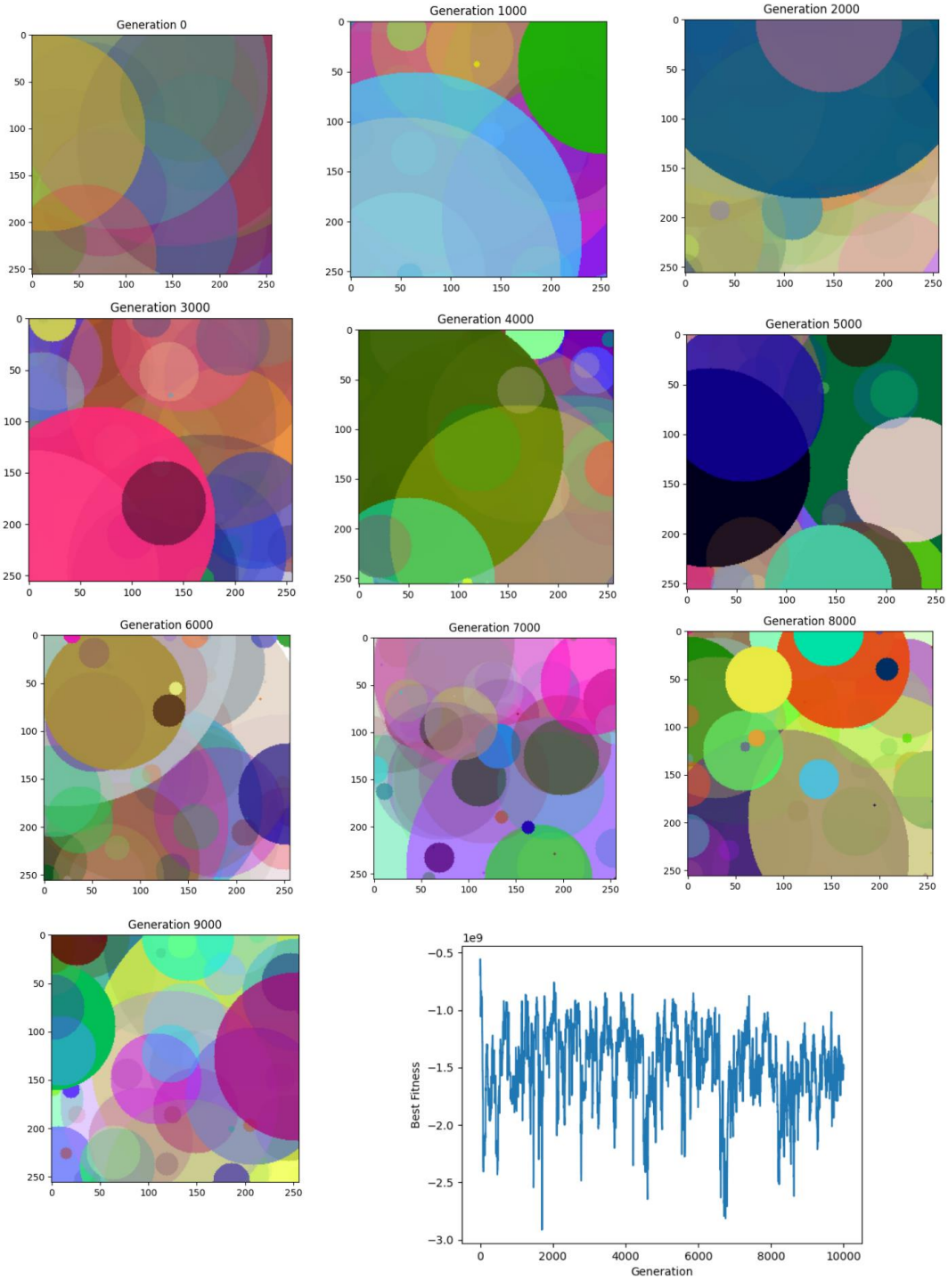
Yasincan Bozkurt
2304202

Number of Genes = 30



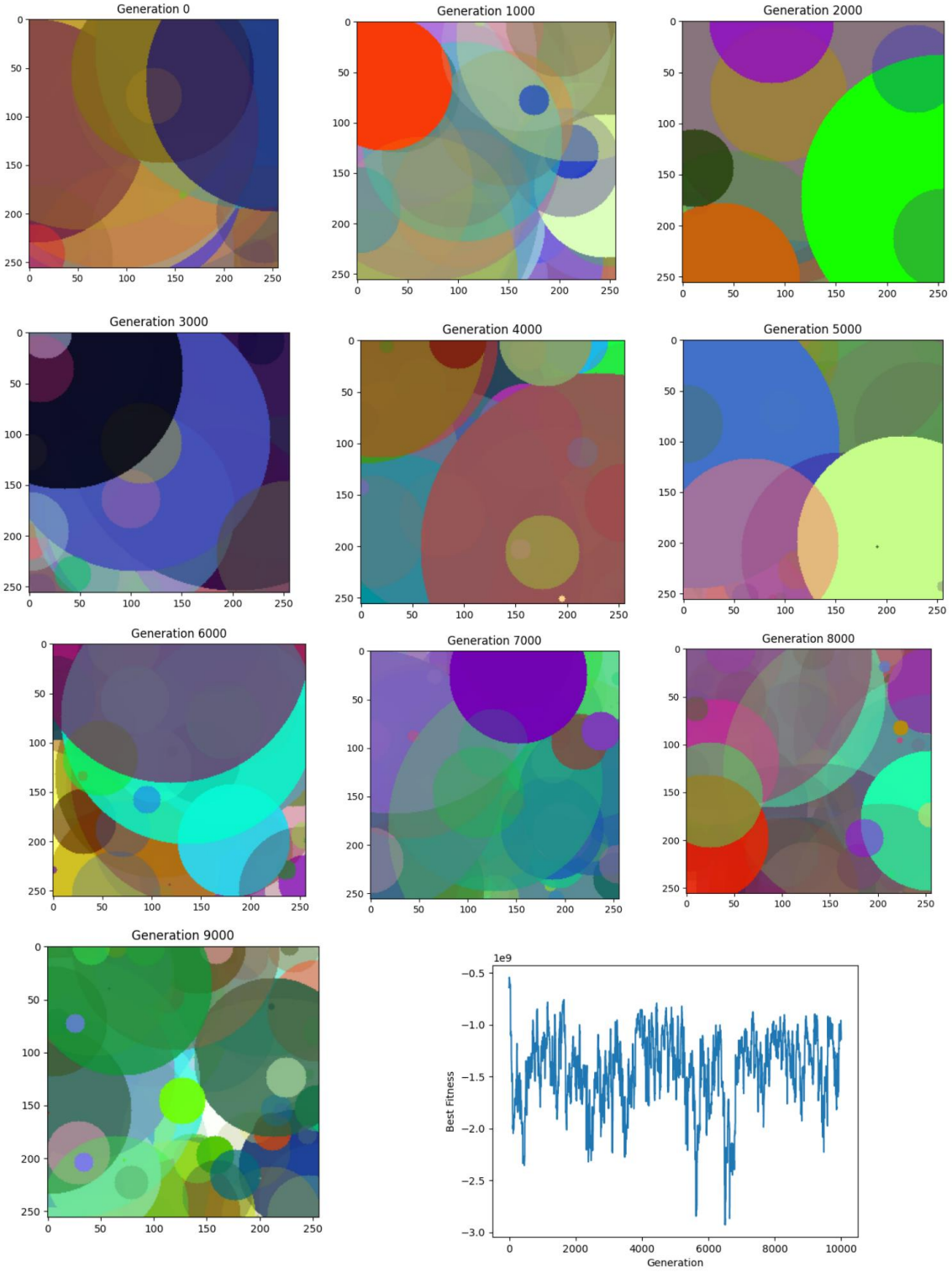
Yasincan Bozkurt
2304202

Number of Genes = 80



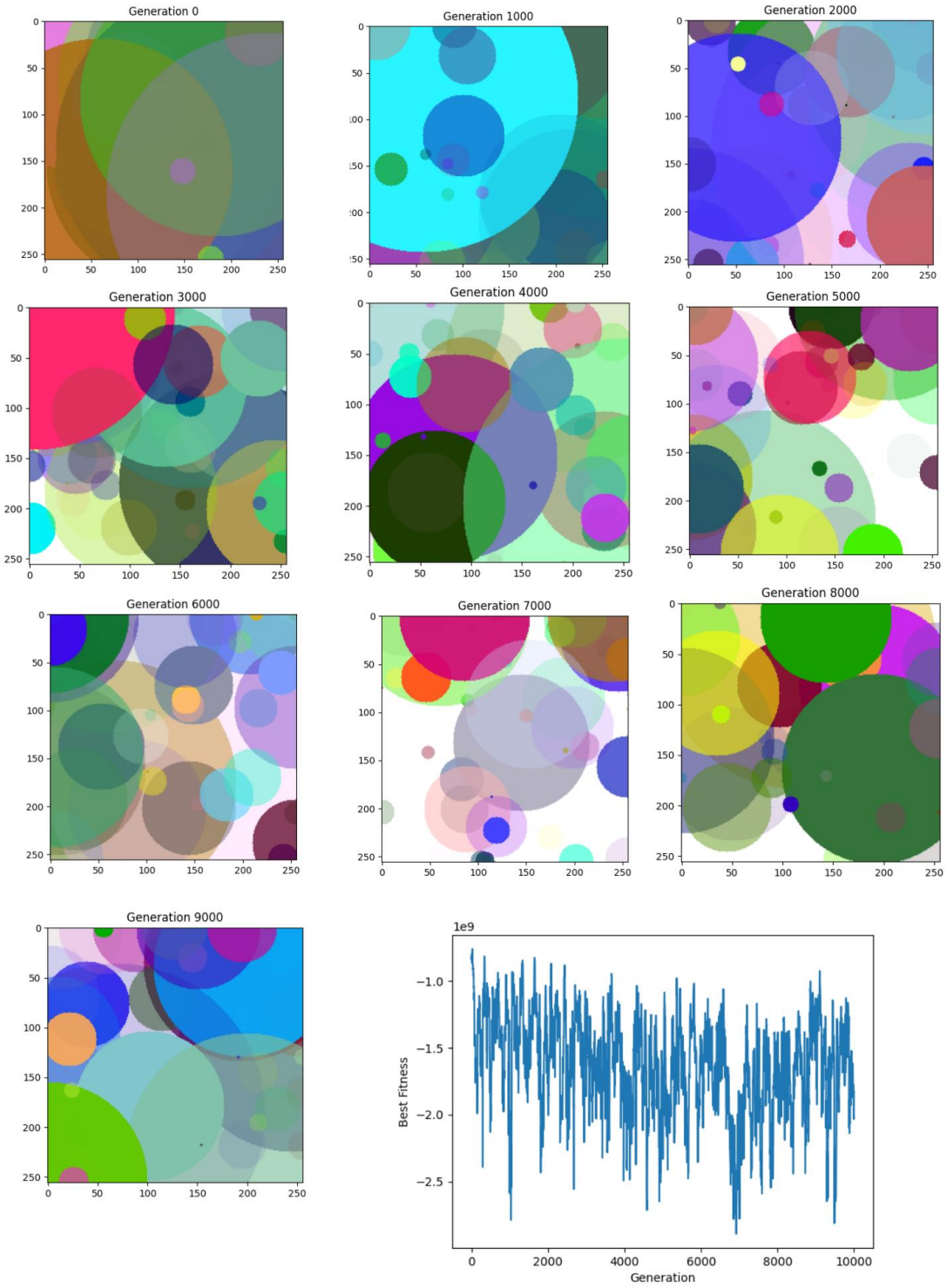
Yasincan Bozkurt
2304202

Number of Genes = 120



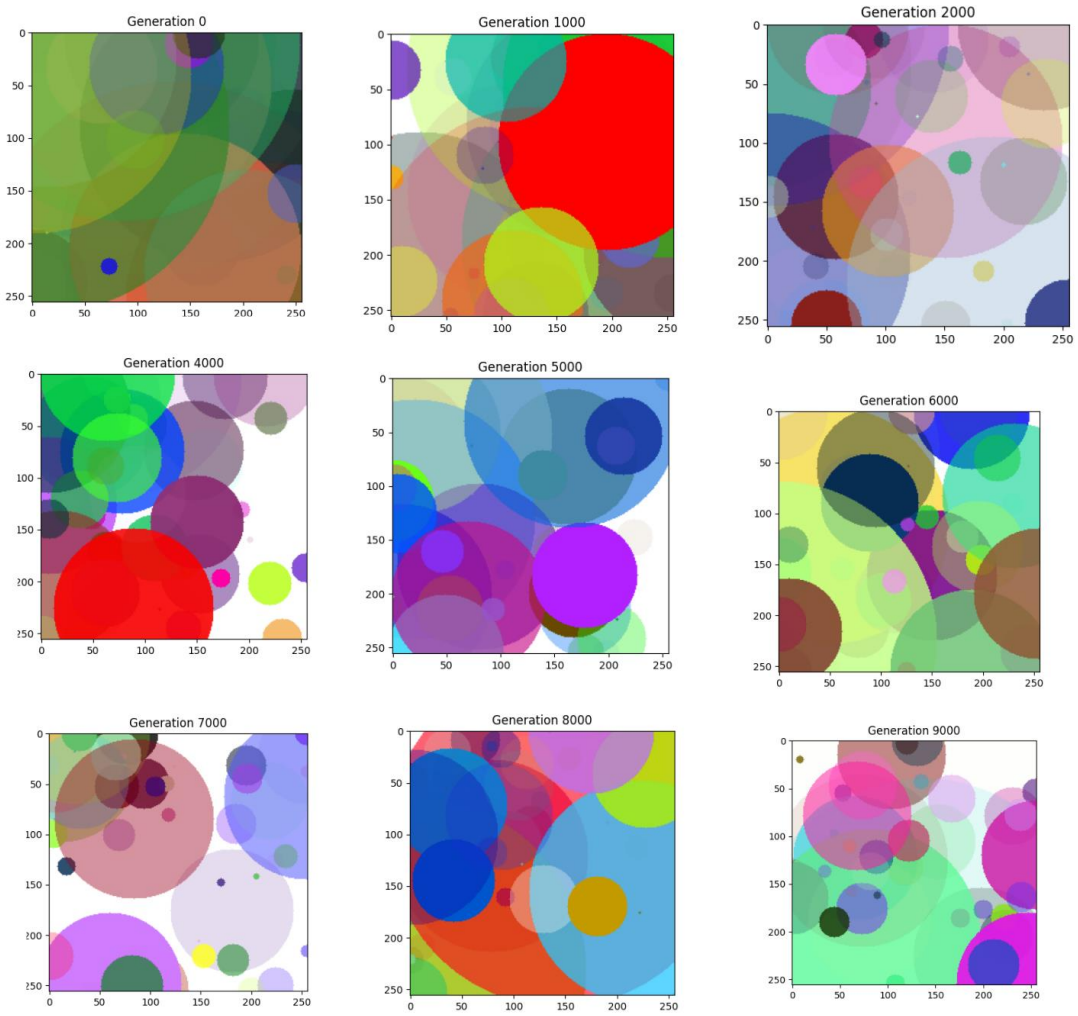
Yasincan Bozkurt
2304202

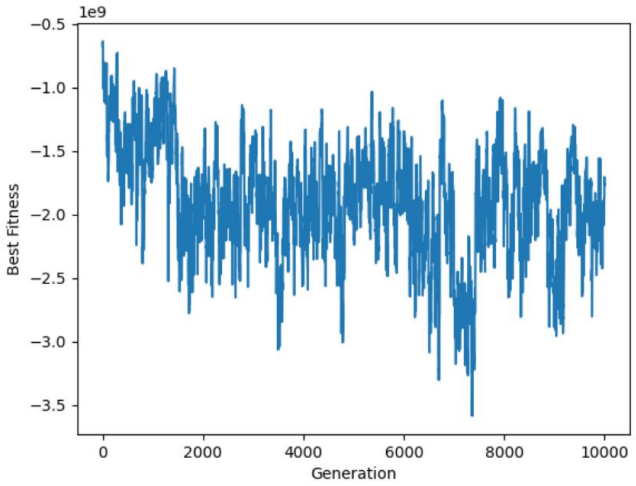
Tournament Size = 2



Yasincan Bozkurt
2304202

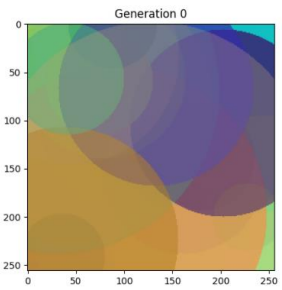
Tournament Size = 8

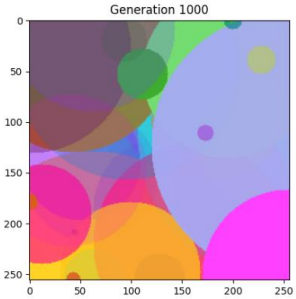


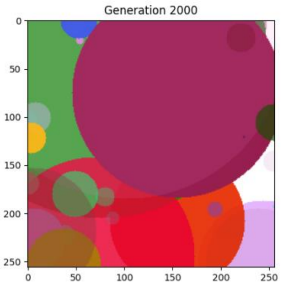


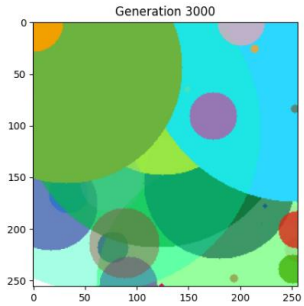
Yasincan Bozkurt
2304202

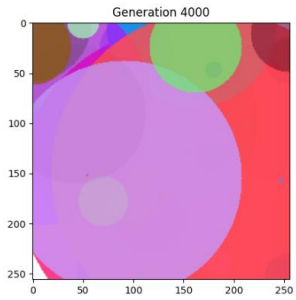
Tournament Size = 16

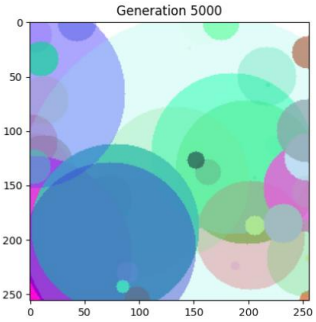


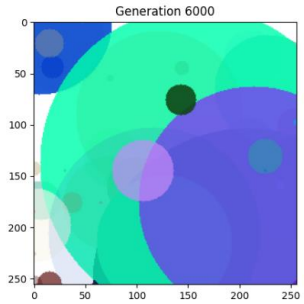


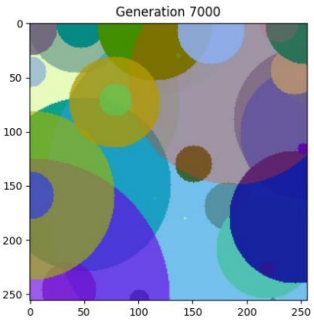


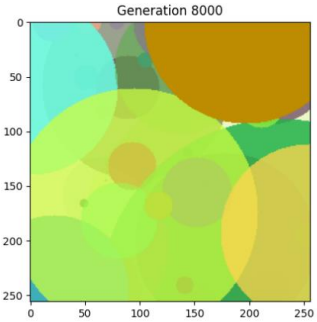


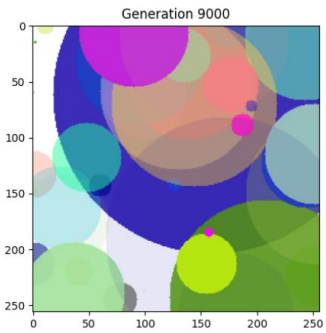


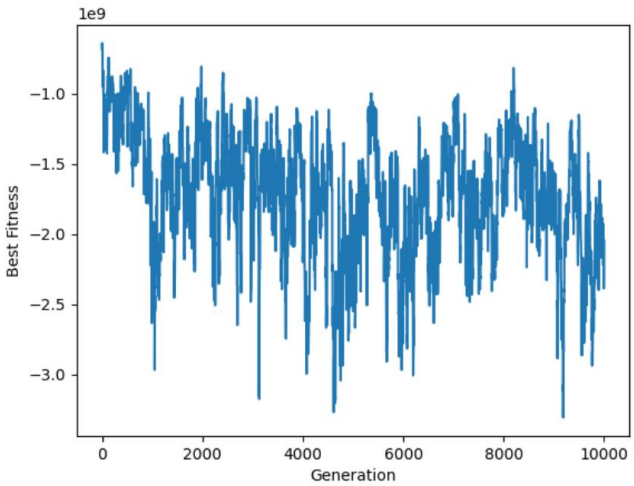






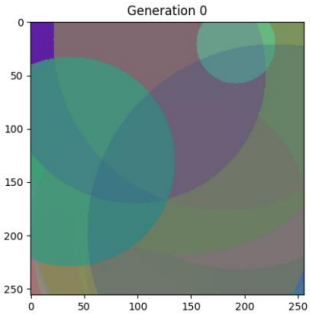


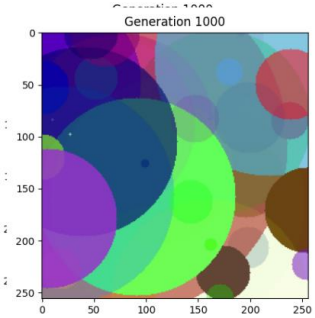


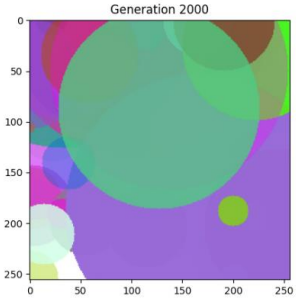


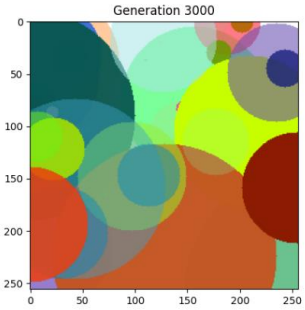
Yasincan Bozkurt
2304202

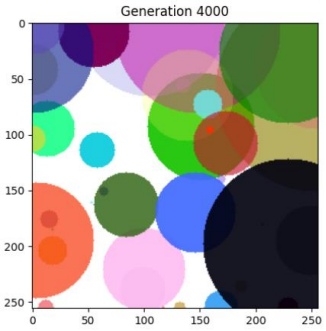
Fraction of Elites = 0.04

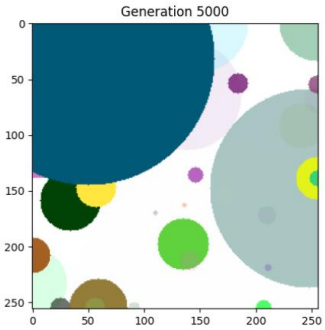


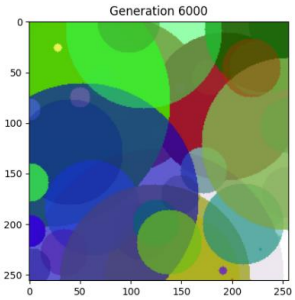


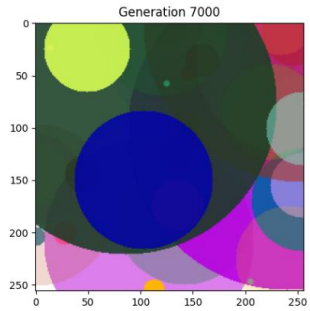


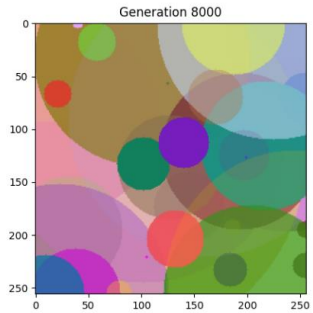


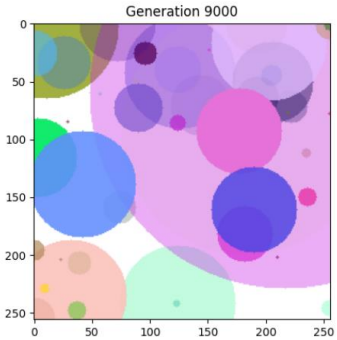


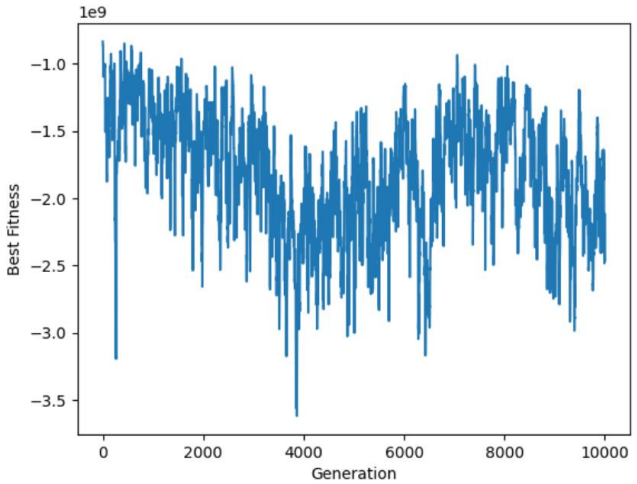






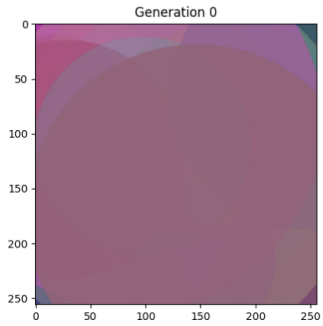


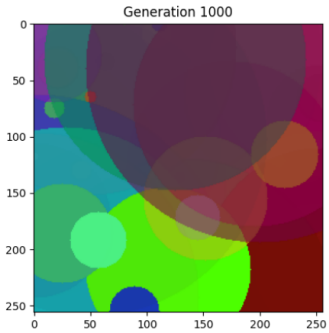


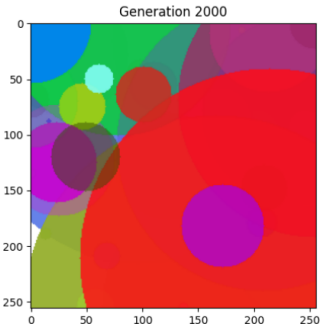


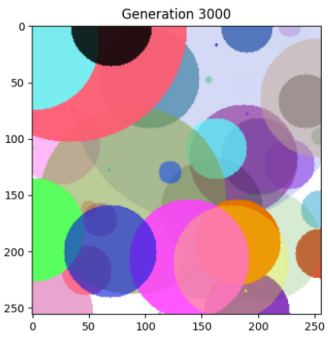
Yasincan Bozkurt
2304202

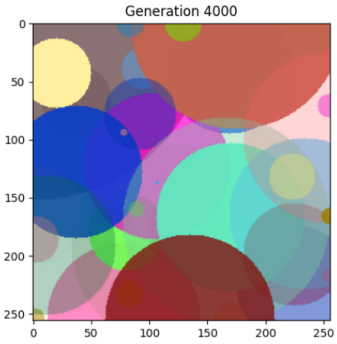
Fraction of Elites = 0.35

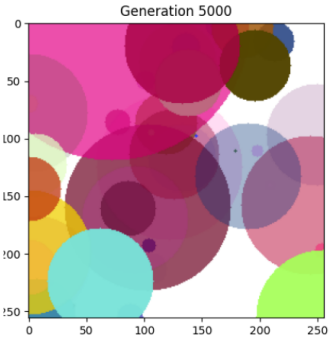


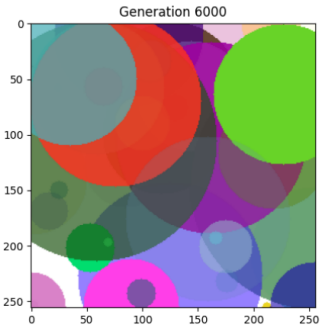


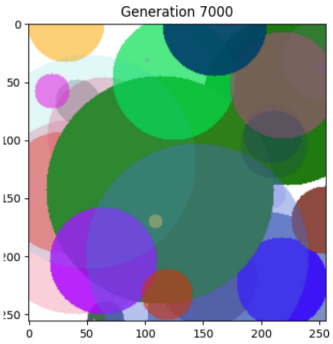


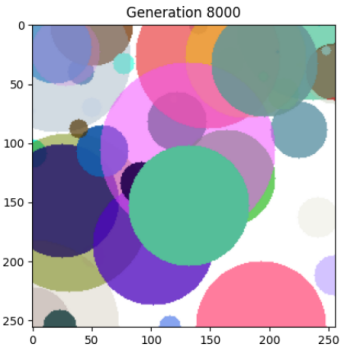


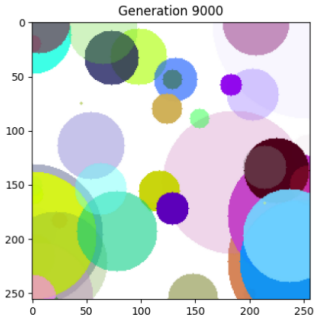


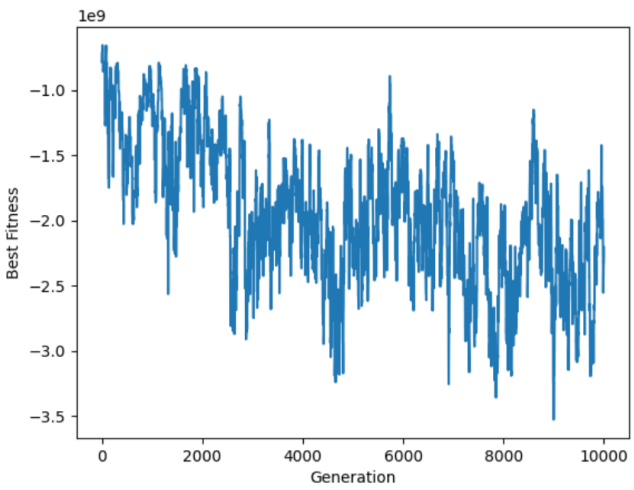






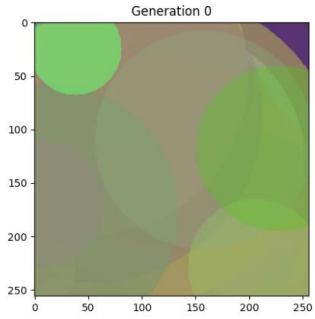


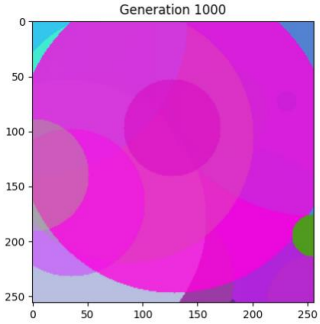


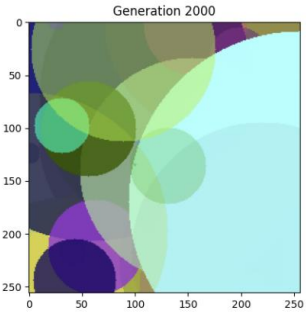


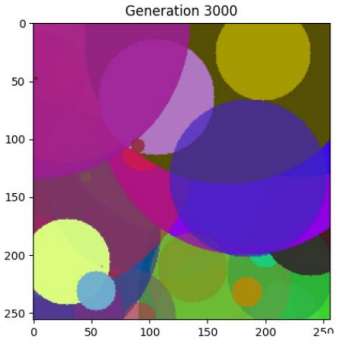
Yasincan Bozkurt
2304202

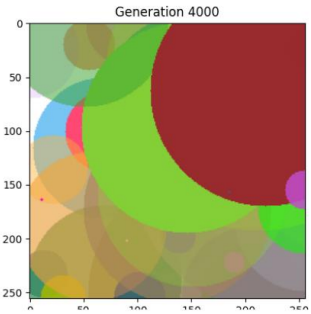
Mutation Probability = 0.1

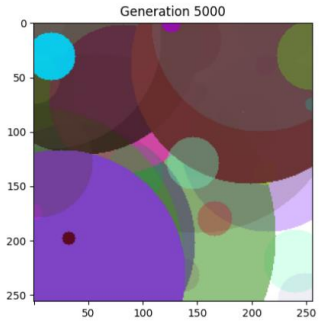


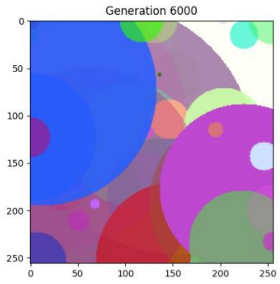


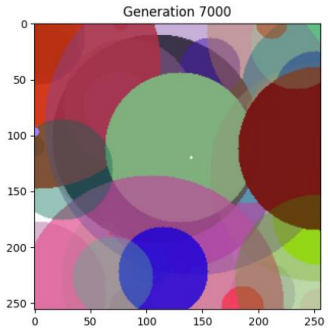


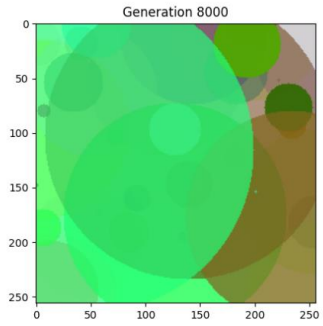


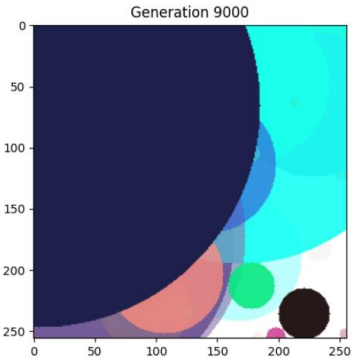


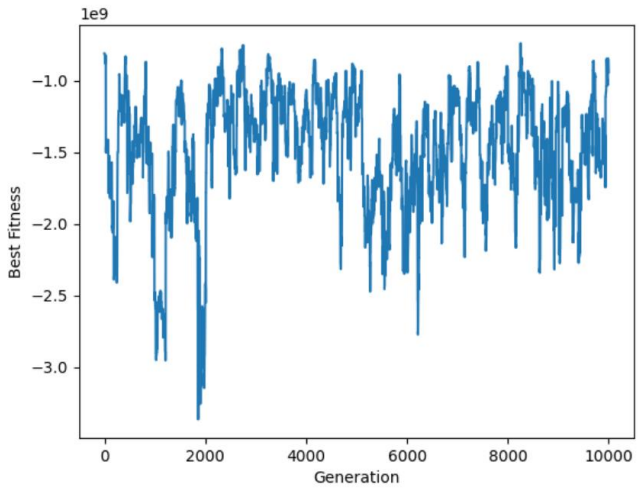






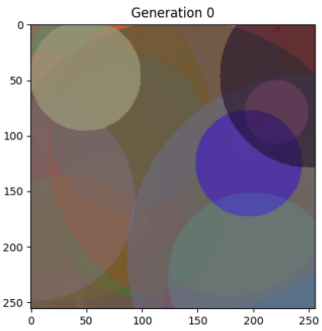


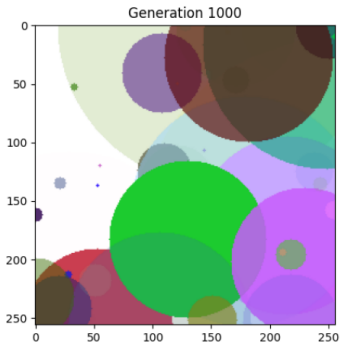


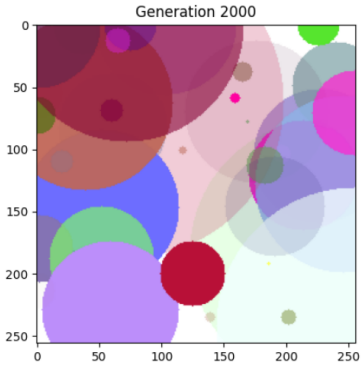


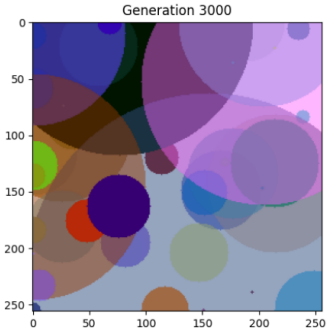
Yasincan Bozkurt
2304202

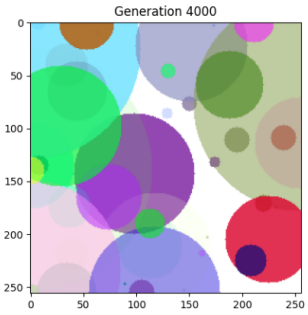
Mutation Probability = 0.4

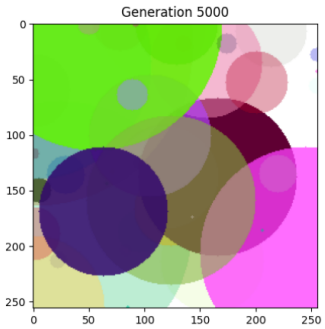


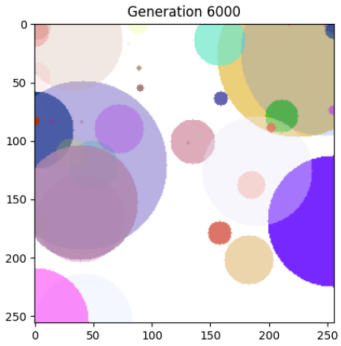


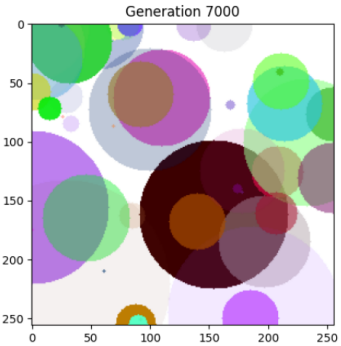


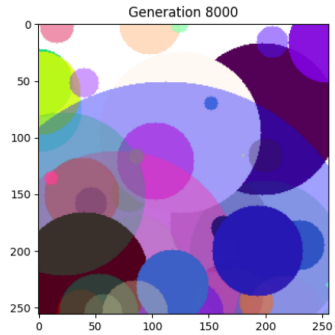


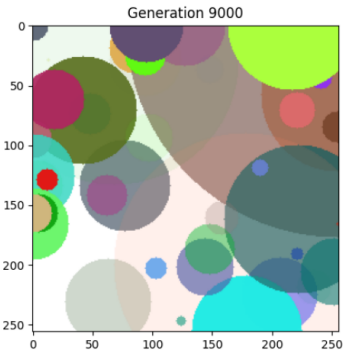


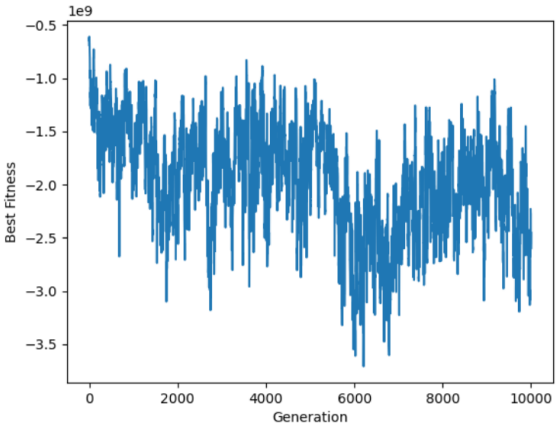






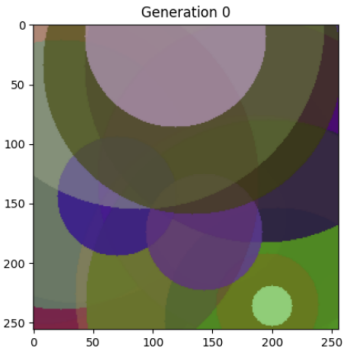


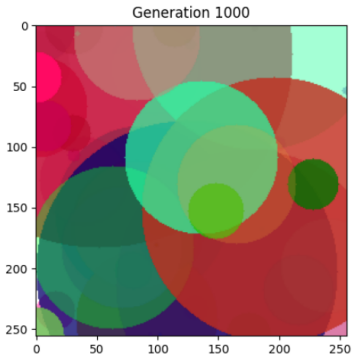


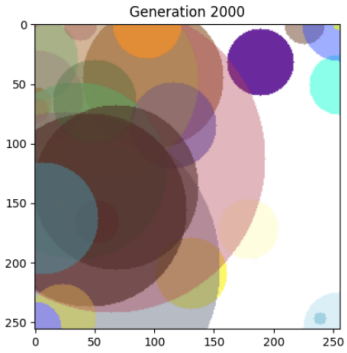


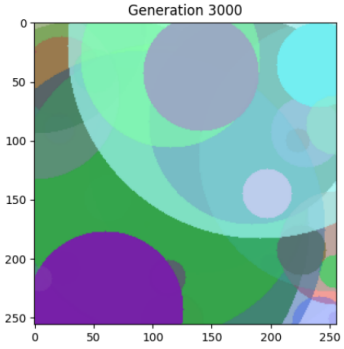
Yasincan Bozkurt
2304202

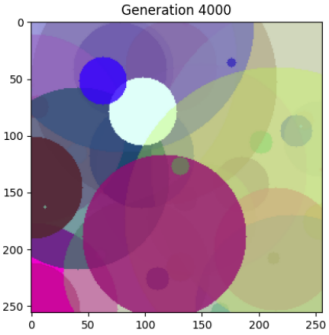
Mutation Probability = 0.75

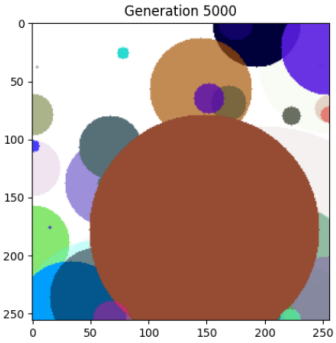


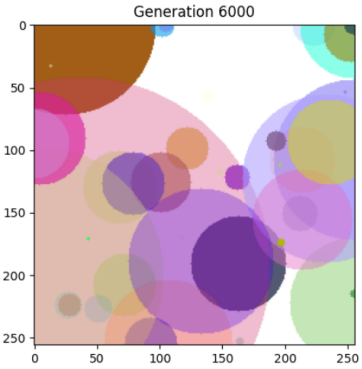


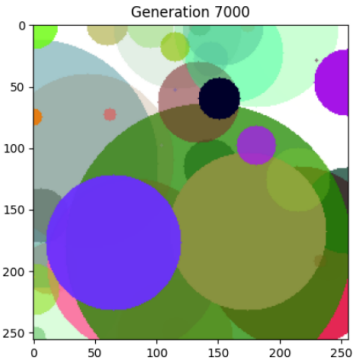


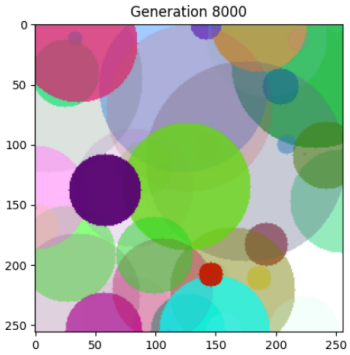


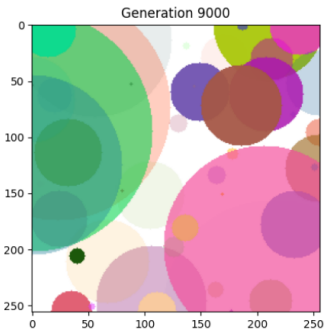


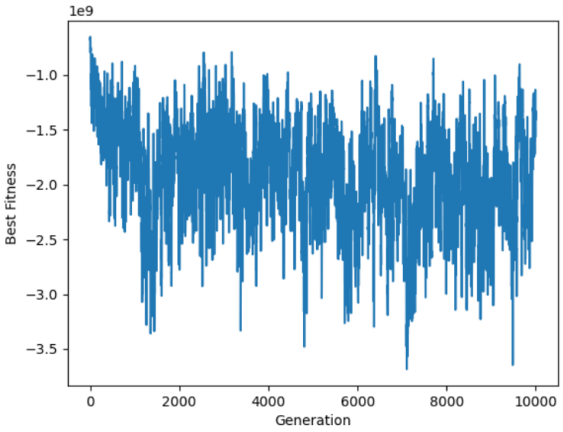






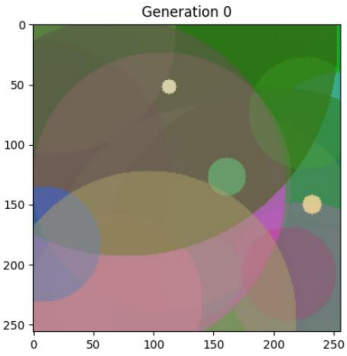


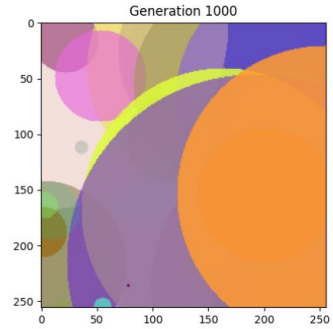


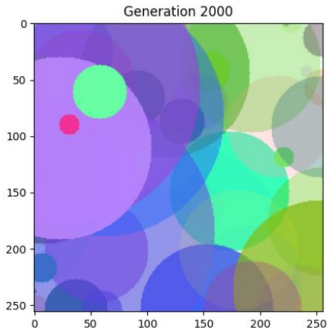


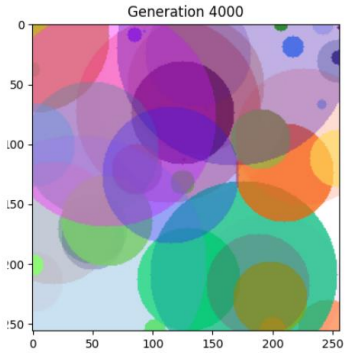
Yasincan Bozkurt
2304202

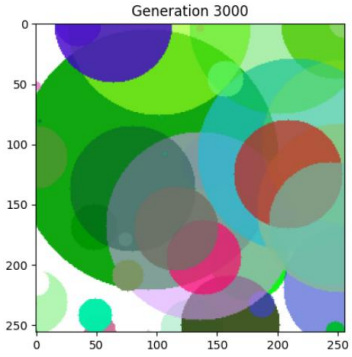
Fraction of Parents = 0.15

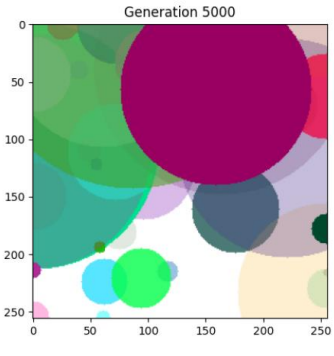


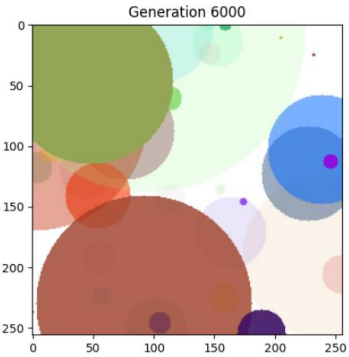


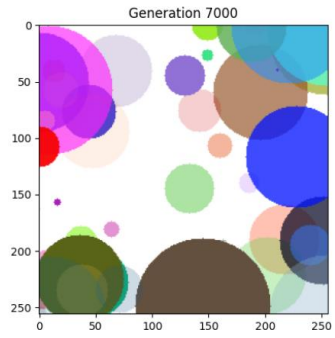


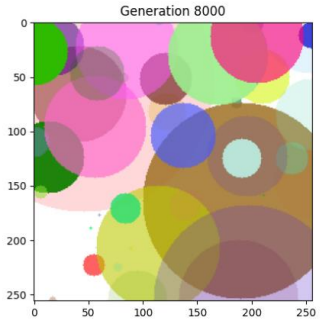


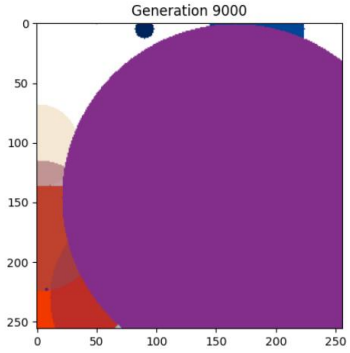


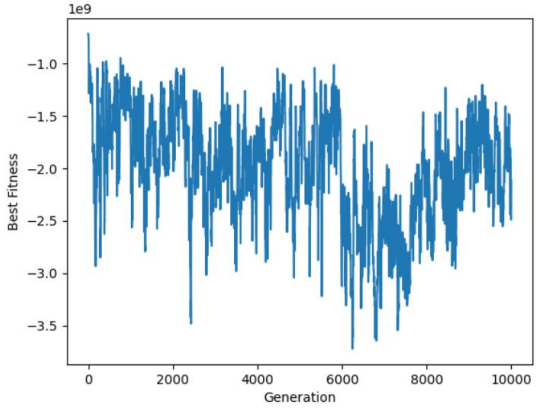






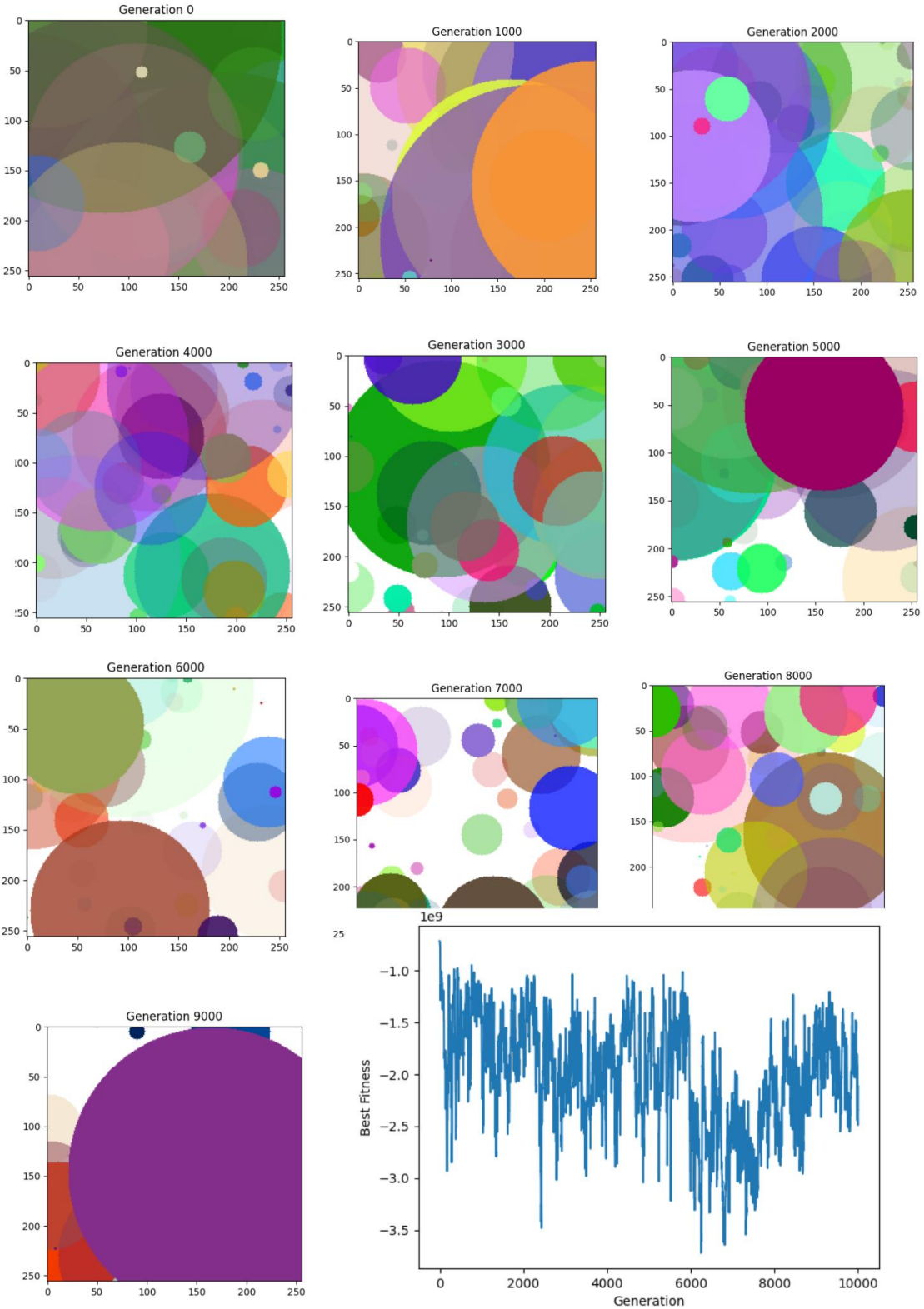






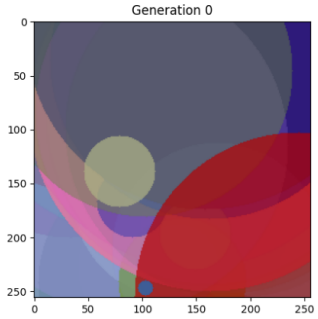
Yasincan Bozkurt
2304202

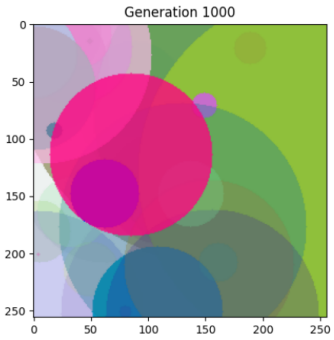
Fraction of Parents = 0.3

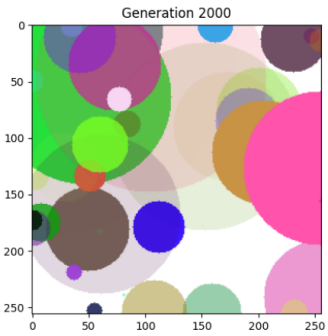


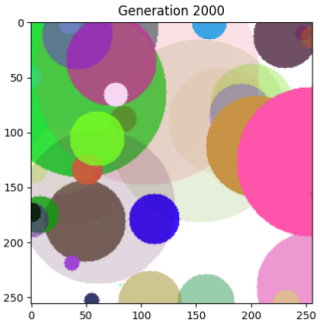
Yasincan Bozkurt
2304202

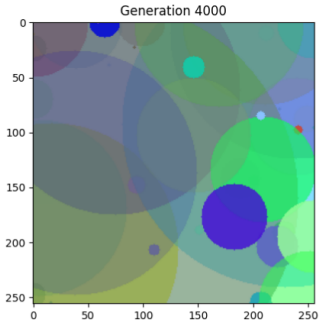
Fraction of Parents = 0.15

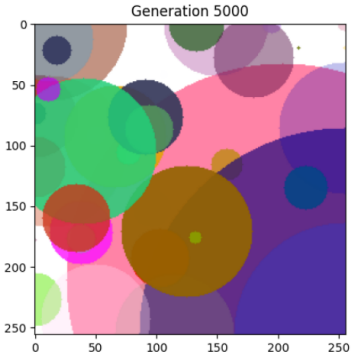


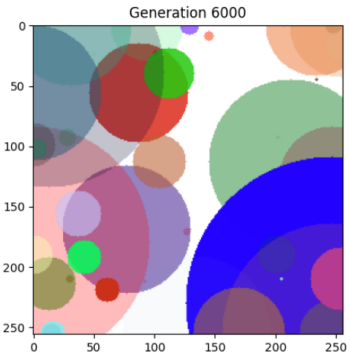


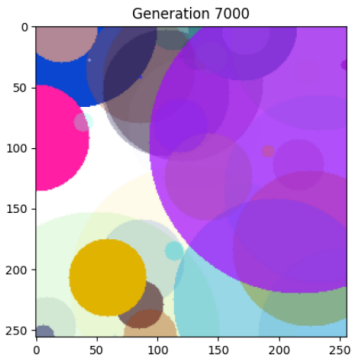


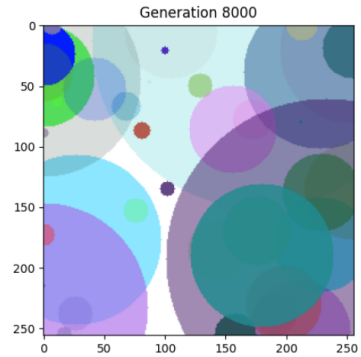


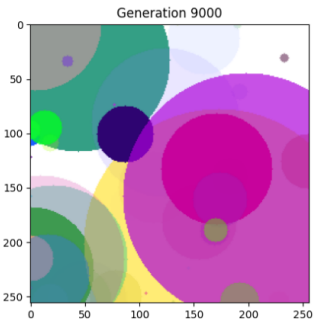


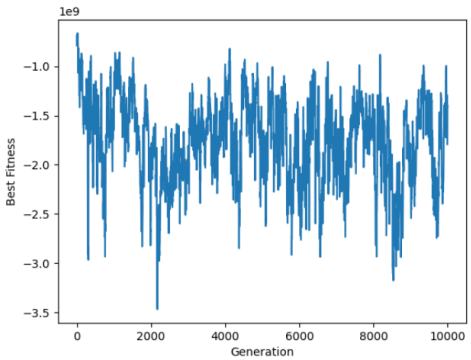






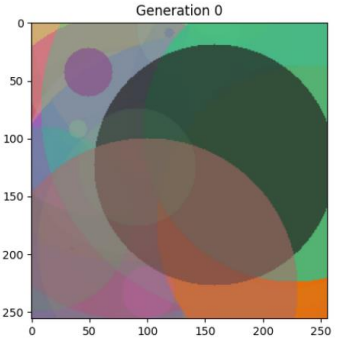


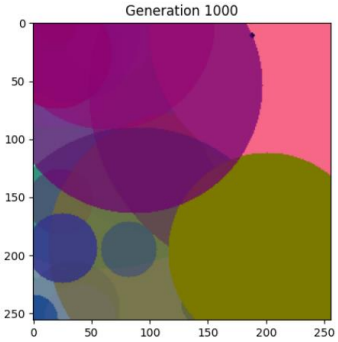


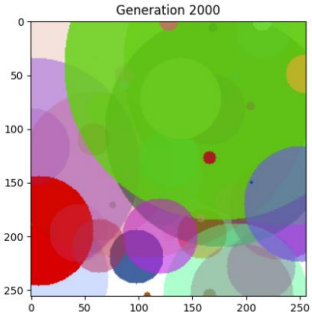


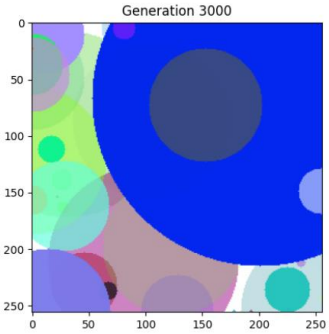
Yasincan Bozkurt
2304202

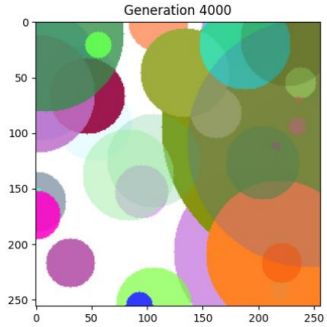
Fraction of Parents = 0.75

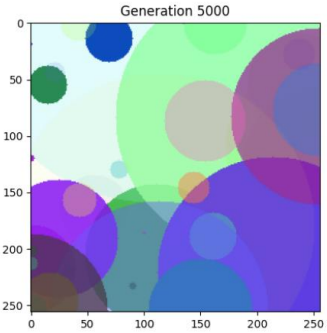


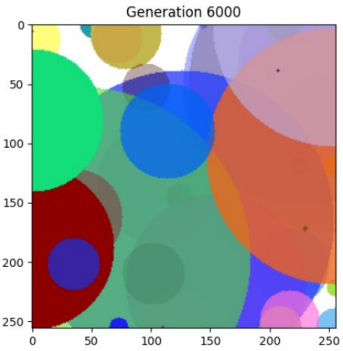


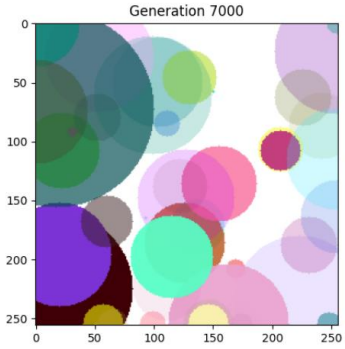


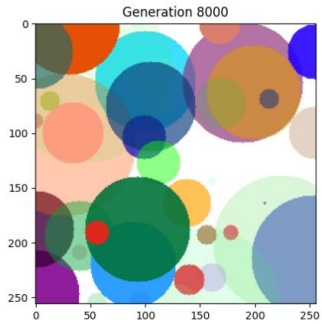


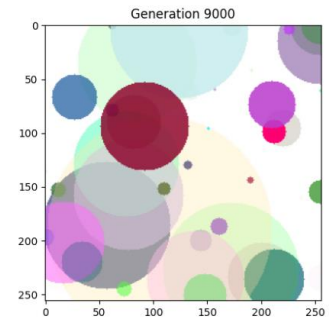


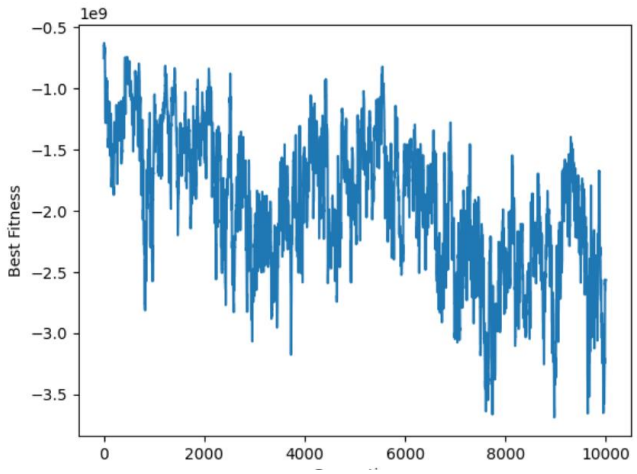












Yasincan Bozkurt
2304202

The code that is used is given. Parameters changed for each round according to table.

```
#import libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import random

# Parameters
#This part is changed with different parameters according to table.
#Bold parts are standart and one variable is changed in each experiment
number_of_individuals = 20 # number of individuals
number_of_genes = 50  # number of genes
num_generation = 10000  # number of generations
tm_size = 5  # tournament size
num_elites = 0.2  # fraction of elites
num_parents = 0.6  # fraction of parents
mutation_prob = 0.75  # mutation probability
mutation_type = "guided"  #guided ungided selections
width, height = 256, 256  # dimension of image
max_radius = int(np.sqrt(width**2 + height**2) / 2)

# Reading source image
#Path for image
#
source_image = cv2.imread(r'C:\Users\yasin\Downloads\painting.png',
cv2.COLOR_BGR2RGB)
source_image = cv2.resize(source_image, (width, height))
#gene classs
class Gene:
    def __init__(self):
        self.y = np.random.randint(0, width)
        self.x = np.random.randint(0, height)
        self.radius = np.random.randint(1, max_radius)
        self.r = np.random.randint(0, 256)
        self.g = np.random.randint(0, 256)
        self.b = np.random.randint(0, 256)
        self.a = np.random.rand()


    def mutate(self):
        if mutation_type == "guided":
            self.x = np.clip(self.x + np.random.randint(-width//4, width//4),
0, width)
            self.y = np.clip(self.y + np.random.randint(-height//4,
height//4), 0, height)
            self.radius = np.clip(self.radius + np.random.randint(-10, 10), 1,
width // 2)
            self.r = np.clip(self.r + np.random.randint(-64, 64), 0, 256)
```

```
            self.g = np.clip(self.g + np.random.randint(-64, 64), 0, 256)
            self.b = np.clip(self.b + np.random.randint(-64, 64), 0, 256)
            self.a = np.clip(self.a + np.random.uniform(-0.25, 0.25), 0, 1)
        else:  # unguided mutation
            #directly use
            self.__init__()
#individial class
class Individual:
    def __init__(self, genes=None):
        if genes is None:
            self.genes = [Gene() for _ in range(number_of_genes)]
        else:
            self.genes = genes
        self.fitness = self.evaluate()

    def evaluate(self):
        # omitted for brevity

        def sort_genes(self):
            self.genes.sort(key=lambda gene: gene.radius, reverse=True)


    def evaluate(self):
        image = np.ones((width, height, 3), np.uint8) * 255
        for gene in self.genes:
            overlay = image.copy()
            cv2.circle(overlay, (gene.x, gene.y), gene.radius, (gene.b,
gene.g, gene.r), -1)
            image = cv2.addWeighted(overlay, gene.a, image, 1 - gene.a, 0)
        self.image = image
        self.fitness = -np.sum((source_image.astype("float") -
image.astype("float")) ** 2)
        return self.fitness
#population class
class Population:
    def __init__(self):
        self.individuals = [Individual() for _ in
range(number_of_individuals)]

    def selection(self):
        self.individuals.sort(key=lambda individual: individual.fitness,
reverse=True)
        next_generation =
self.individuals[:int(num_elites*number_of_individuals)]
        for _ in range(int(number_of_individuals -
num_elites*number_of_individuals)):
            tournament = np.random.choice(self.individuals, tm_size)
            tournament = sorted(tournament, key=lambda individual:
individual.fitness, reverse=True)
```

```
            next_generation.append(tournament[0])
        self.individuals = next_generation


    def crossover(self):
        num_cross = int((number_of_individuals -
num_elites*number_of_individuals)/2)
        for _ in range(num_cross):
            # select parents
            parents =
random.sample(self.individuals[int(num_elites*number_of_individuals):int((num_
elites+num_parents)*number_of_individuals)], 2)
            for i in range(2):
                # create child
                child_genes = [parents[np.random.randint(0, 2)].genes[j] for j
in range(number_of_genes)]
                child = Individual(child_genes)
                # add child to population
                self.individuals.append(child)


    def mutation(self):
        for individual in
self.individuals[int(num_elites*number_of_individuals):]:
            if np.random.rand() < mutation_prob:
                gene = individual.genes[np.random.randint(0, number_of_genes)]
                if mutation_type == "unguided":
                    gene.x = np.random.randint(0, width)
                    gene.y = np.random.randint(0, height)
                    gene.radius = np.random.randint(1, max_radius)
                    gene.r = np.random.randint(0, 256)
                    gene.g = np.random.randint(0, 256)
                    gene.b = np.random.randint(0, 256)
                    gene.a = np.random.rand()
                elif mutation_type == "guided":
                    gene.x = min(max(0, gene.x + np.random.randint(-width//4,
width//4)), width-1)
                    gene.y = min(max(0, gene.y + np.random.randint(-height//4,
height//4)), height-1)
                    gene.radius = min(max(1, gene.radius + np.random.randint(-
10, 10)), max_radius)
                    gene.r = min(max(0, gene.r + np.random.randint(-64, 64)),
255)
                    gene.g = min(max(0, gene.g + np.random.randint(-64, 64)),
255)
                    gene.b = min(max(0, gene.b + np.random.randint(-64, 64)),
255)
                    gene.a = min(max(0, gene.a + (np.random.rand()-0.5)/2), 1)
                individual.fitness = individual.evaluate()
```

```
# Run genetic algorithm
population = Population()
best_individuals = []
best_fitnesses = []
for generation in range(num_generation):
    population.selection()
    population.crossover()
    population.mutation()
    population.individuals.sort(key=lambda individual: individual.fitness,
reverse=True)
    best_fitnesses.append(population.individuals[0].fitness)
    # For each 1000 generation
    if generation % 1000 == 0:
        best_individuals.append(population.individuals[0])
        print(f"Generation {generation}")

# Plot fitness
plt.plot(best_fitnesses)
plt.xlabel("Generation")
plt.ylabel("Best Fitness")
plt.show()

# Display image as a result
for i, individual in enumerate(best_individuals):
    plt.imshow(cv2.cvtColor(individual.image, cv2.COLOR_BGR2RGB))
    plt.title(f"Generation {i*1000}")
    plt.show()
#Yasincan Bozkurt
#2304202
```