Maryna Kolesnykova

Student_ID: 122475

Zen garden (genetic algorithm)

A Zen Garden is an area filled with coarser sand (small pebbles). However, it also contains immovable larger objects, such as stones, statues, constructions, self-growths. The monk is supposed to adjust the sand in the garden with a rake so that strips like the one in the following picture are formed. Belts can only go horizontally or vertically, never diagonally. It always starts at the edge of the garden and pulls a straight strip to the other edge or to an obstacle. On the edge - outside the garden, he can walk as he wants. However, if he comes to an obstacle - a stone or already buried sand - he must turn around if he has room. If he has free directions left and right, it is up to him where he turns. If he has only one free direction, he turns there. If he has nowhere to turn, it's game over. A successful game is one in which the monk can dig up the entire garden under the given rules, in this case the maximum possible number of squares. The output is the coverage of the given garden by the transitions of the monk.

Contant

Genetic Algorithm..1

Monk`s movement and behavior..6

New generations (creation methods)...7

Testing...8

Conclusion.. 19

Genetic Algorithm

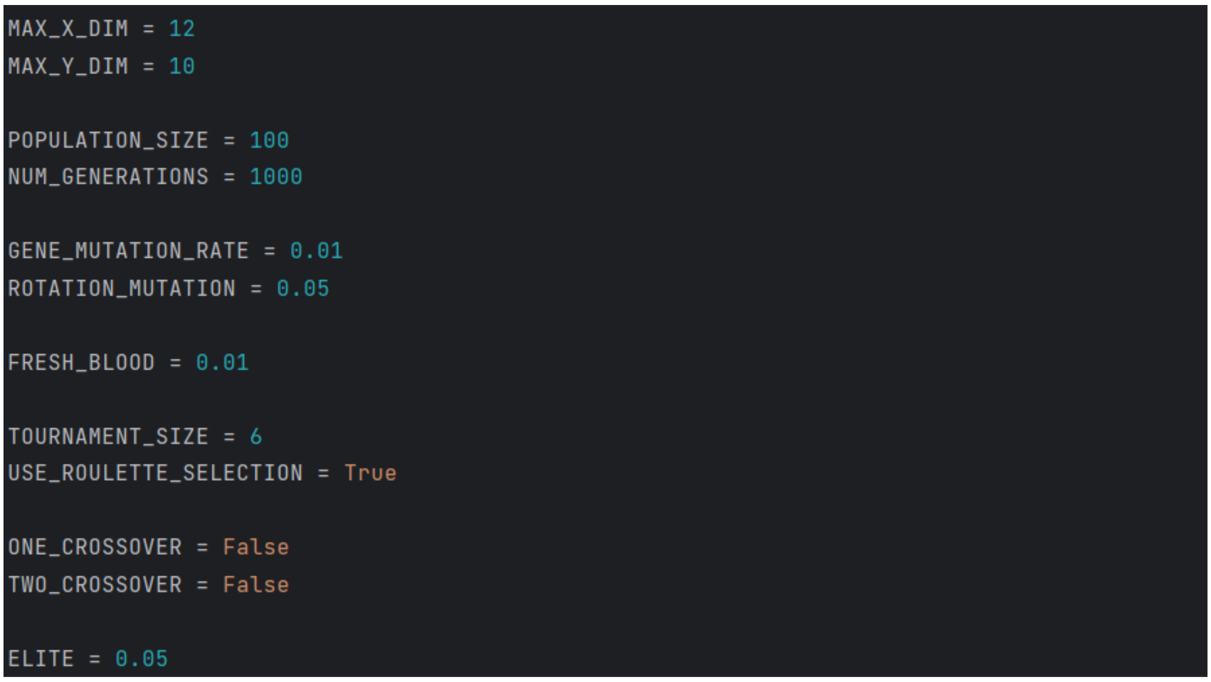
The most important part of the solution is usage of genetic algorithm. Genetic algorithm is a search and optimization technique inspired by the process of natural selection and genetics. It is used to find approximate solutions to complex optimization and search problems.

Genetic algorithms are particularly useful in cases where traditional methods may be less effective or efficient.
Genetic algorithm has several crucial aspects that are listed below:

- Population: A genetic algorithm starts with a population of potential solutions (individuals). Each individual represents a possible solution to the problem. The population is evolved over generations.
- Chromosome: Each individual in the population has a chromosome, which is a data structure that encodes a potential solution. In many cases, the chromosome is represented as a binary string, but it can take various forms depending on the problem.
- Genes: The individual elements within a chromosome are called genes. Genes represent specific parts or characteristics of the solution.
- Fitness Function: A fitness function is defined to evaluate the quality of each individual's solution. It quantifies how close an individual's solution is to the optimal solution. The higher the fitness value, the better the solution.
- Selection: Selection is the process of choosing individuals from the current population to serve as parents for the next generation. Individuals with higher fitness values are more likely to be selected. Various selection methods, such as roulette wheel selection and tournament selection, can be used.
- Crossover (Recombination): Crossover is the process of combining genetic information from two parents to create one or more offspring. The offspring inherit genes from both parents. Different crossover techniques, such as one-point crossover and two-point crossover, can be used.
- Mutation: Mutation introduces small random changes into an individual's chromosome. This helps introduce diversity into the population and prevent premature convergence to a suboptimal solution.
- Elitism: Elitism involves preserving a certain percentage of the best-performing individuals from one generation to the next without any changes. This ensures that the best solutions are not lost during evolution.

Termination Criteria: Genetic algorithms run for a specified number of generations or until a termination condition is met. Common termination conditions include a maximum number of generations, a solution with satisfactory fitness, or a fixed time limit.

All of these parameters can be changed in the code and are represented as global parameters at the beginning of the code(can be int or Boolean)



```
MAX_X_DIM = 12
MAX_Y_DIM = 10

POPULATION_SIZE = 100
NUM_GENERATIONS = 1000

GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.05

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = False

ELITE = 0.05
```

MAX_X_DIM and MAX_Y_DIM are width and height of the field (length of module x and module y coordinates)

POPULATION_SIZE is number of monks in one population

NUM_GENERATIONS is maximum number of generations that can be used to find the solution(in best cases the limit will not be reached, this parameters is a limit that is used to control run time of the program)

GENE_MUTATION_RATE is a parameter that represents the mutation rate of gene mutations

ROTATION_MUTATION is a parameter that represents the mutation rate of rotation mutations

FRESH_BLOOD the rate at which fresh genetic material is introduced into the population

TOURNAMENT_SIZE the size of tournament selection pool

USE_ROULETTE_SELECTION a Boolean flag to choose between roulette wheel selection and tournament selection (if it is true roulette selection will be performed. Another way the program will use tournament selection)

ONE_CROSSOVER and TWO CRESSOVER are flags to enable or disable crossover methods.

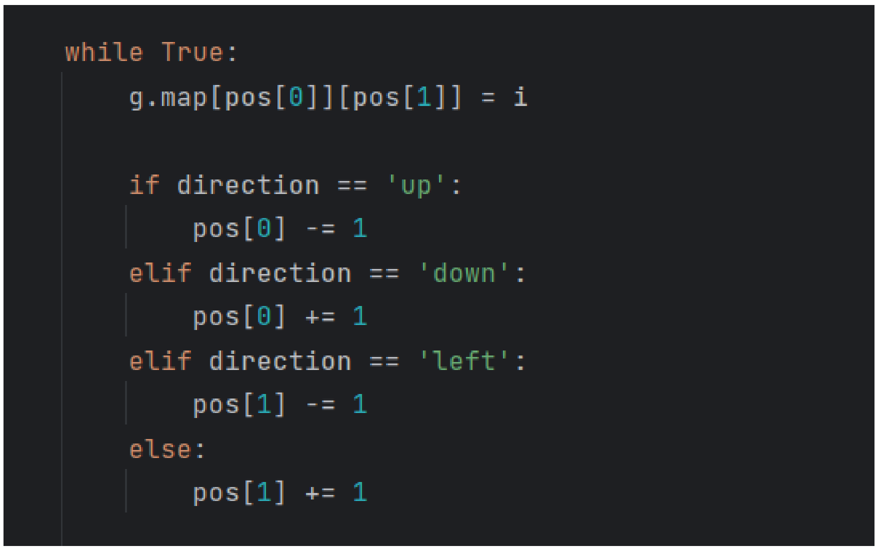
ELITE the proportion of elite individuals to preserve in each generation.

Description of genes

1. Gene Representation
   A gene represents a specific action that instructs how a garden plot should be placed within the grid. Genes have the following attributes:
   start: This attribute specifies the starting position for the garden plot, represented as a tuple containing the (x, y) coordinates within the garden grid.
   direction: This attribute indicates the direction in which the garden plot should be moved, and it can take one of the following values: 'up,' 'down,' 'left,' or 'right.'
   rotation: The rotation attribute is a list of binary values (0 or 1), representing a rotation sequence that dictates how the garden plot should rotate during placement.

2. Gene Initialization
   Genes are created and initialized within the GeneticPathFinder class. When a new genetic individual is created, genes are generated, and each gene is initialized with a random starting position and direction. Additionally, each gene's rotation sequence is generated and initialized randomly. The creation of a gene involves the following steps:

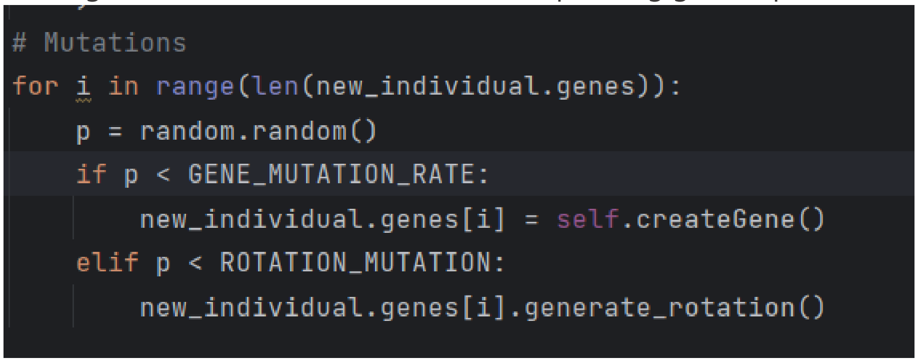
   Randomly select a starting position within the garden grid.
   Randomly assign one of the four possible directions: 'up,' 'down,' 'left,' or 'right' for the movement of the garden plot.



```
while True:
    g.map[pos[0]][pos[1]] = i

    if direction == 'up':
        pos[0] -= 1
    elif direction == 'down':
        pos[0] += 1
    elif direction == 'left':
        pos[1] -= 1
    else:
        pos[1] += 1
```

Generate a random rotation sequence consisting of 10 binary values (0 or 1) that define the rotation behavior of the plot.

3. Gene Mutaion
Mutation is a key element in the genetic algorithm and plays a role in introducing variability into the gene sequences. Specifically, two types of mutations are applied to the genes:
Gene Mutation: Gene mutation occurs with a probability specified by the GENE_MUTATION_RATE parameter. When mutation occurs, a gene is completely replaced with a new randomly generated gene. This introduces diversity into the population and can help in exploring new solutions.
Rotation Mutation: Rotation mutation is applied with a probability specified by the ROTATION_MUTATION parameter. This mutation affects the rotation sequence of the genes. It randomly changes some of the binary values in the rotation sequence, altering the rotation behavior of the corresponding garden plot.



```
# Mutations
for i in range(len(new_individual.genes)):
    p = random.random()
    if p < GENE_MUTATION_RATE:
        new_individual.genes[i] = self.createGene()
    elif p < ROTATION_MUTATION:
        new_individual.genes[i].generate_rotation()
```

4. Gene usage in solution
Once genes are generated and initialized, they are used to create a sequence of actions for placing garden plots in the garden grid. The solution method in the GeneticPathFinder class iterates through the genes and uses their information to determine how and where each garden plot should be placed within the grid. The goal is to maximize the coverage of empty cells while avoiding rocks. The garden

grid is modified according to the actions specified by the genes. The direction attribute of each gene guides the movement of the garden plot in the specified direction. The rotation sequence (rotation) of the gene is used to determine when and how the garden plot should rotate during placement.

5. Role in evolution
Genes are a fundamental part of the genetic algorithm's process. They are inherited and modified during the crossover and mutation operations. These genetic operations help evolve the population over multiple generations, searching for a combination of genes that will lead to a solution where the garden plots cover the maximum number of empty cells without encountering rocks. In summary, genes in the Garden Puzzle Solver represent the instructions for placing garden plots within the grid. Through mutation and crossover operations, the genetic algorithm explores different combinations of genes to find an optimal solution to the garden puzzle

Monk’s movement and behavior

1. Monk`s genes
The Monk's behavior is encoded using a set of genes, which are represented as instances of the Gene class. Each gene encapsulates the following attributes: start: This attribute specifies the starting position for a garden plot, represented as a tuple with (x, y) coordinates within the garden grid. direction: The direction attribute indicates the movement direction in which the Monk should place the garden plot. It can be one of the following values: 'up,' 'down,' 'left,' or ' right.' rotation: The rotation attribute is a list of binary values (0 or 1) that defines the rotation sequence for the garden plot. This sequence dictates when and how the garden plot should rotate during placement.

2. Monk`s decision-making
The Monk makes decisions about where and how to place garden plots by following a sequence of actions defined by its genes. Here's how the Monk decides its movements:
Initialization: When a Monk is created, it is equipped with a set of genes. These genes are randomly generated and dictate the initial placement and movement behavior of the Monk.
Solution Sequence: The Monk executes a sequence of actions defined by its genes to arrange the garden plots in the garden grid. The sequence is executed one gene at a time, and the Monk follows the instructions provided by each gene.
Navigation: The Monk navigates the garden grid based on the direction specified in each gene. The direction determines whether the Monk should move 'up,' 'down,' 'left,' or 'right' in the grid.

Rotation Sequence: The rotation attribute of the gene dictates when and how the Monk should rotate the garden plot during placement. This rotation sequence ensures that the Monk can adapt to the shape and available space in the garden grid.
Collision Avoidance: The Monk is programmed to avoid collisions with rocks in the grid. If it encounters a rock during its movement, it adjusts its path or rotation to avoid the obstacle and continue its placement strategy.

New generations (creation methods)

1. Initialization
   When a new generation is created, it starts with the existing population of Monk individuals from the previous generation. The Monk individuals are initially equipped with a set of genes that dictate their behavior and movement within the garden.
2. Selection
   The genetic algorithm evaluates the fitness of each Monk in the population based on how effectively they have covered empty cells in the garden while avoiding rocks. Monk individuals with higher fitness scores are more likely to be selected for the next generation.
3. Elitism
   A portion of the best-performing Monk individuals, often referred to as "elite" individuals, is preserved without modification and carried over to the next generation. The number of elite individuals is determined by the ELITE parameter.
4. Fresh blood
   A small percentage of new, randomly generated Monk individuals are introduced to the population. These individuals bring diversity and new genetic material to the population, aiding exploration.
5. Crossover
   Monk individuals are selected from the current population to act as parents for generating new offspring. Two parents are chosen to create one or more offspring. The crossover operation combines the genes of the parents to create new Monk individuals with a mix of genetic traits.
6. Mutation
   Random changes are introduced into the genes of Monk individuals, mimicking genetic mutations. Mutation can include altering the properties of genes, such as changing the starting position or rotation instructions.

7. Solution evaluation
   After creating new Monk individuals, each Monk executes its sequence of actions based on its genes, trying to solve the garden puzzle. The fitness of each Monk is reevaluated to determine how well it covers empty cells in the garden grid.

8. Replacement
   The new generation replaces the previous generation of Monks. Monk individuals from the current generation are replaced with the newly generated Monk individuals while preserving elite individuals.

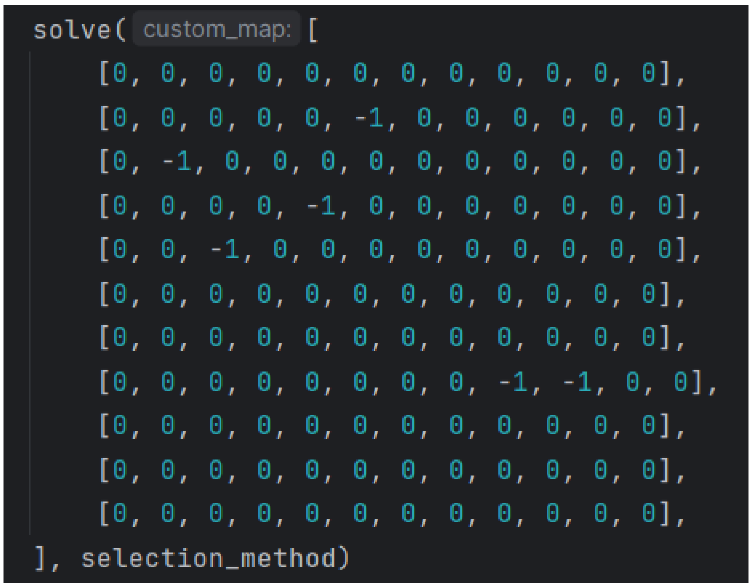
9. Termination
   The process of creating new generations and evaluating their fitness is repeated for multiple generations (as determined by NUM_GENERATIONS). The genetic algorithm continues to evolve the population until a stopping criterion is met. In this implementation, the algorithm terminates when the best Monk covers all the empty cells in the garden.

10. Display and reposting
    During the process, information about the generations and the fitness of the best Monk is displayed. The final solution, garden layout, and details about the best-performing Monk are reported when the algorithm completes or reaches the stopping criterion.

Testing

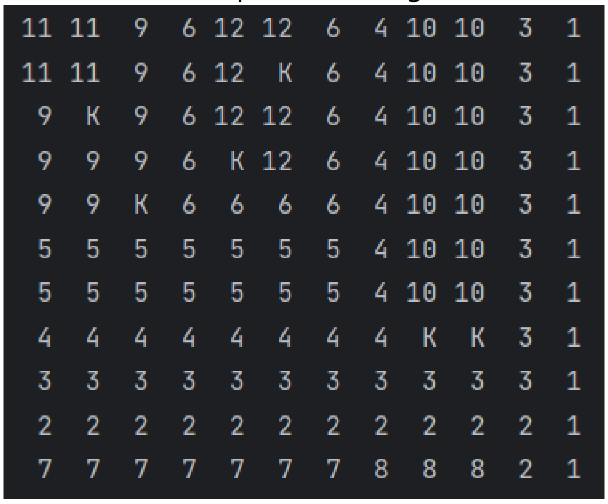
This section provides an overview of the testing process conducted to validate the performance and functionality of the genetic algorithm program developed for solving the garden puzzle

The testing will be provided for the same matrix (testing matrix pic 1) for several changeable parameters. The result of the testing is a graph representing maximum and average fitness. For the clear testing population size will always be 100 and max number of generations will be 1000.



```
solve( custom_map: [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0],
    [0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
], selection_method)
```

pic 1 – Testing matrix

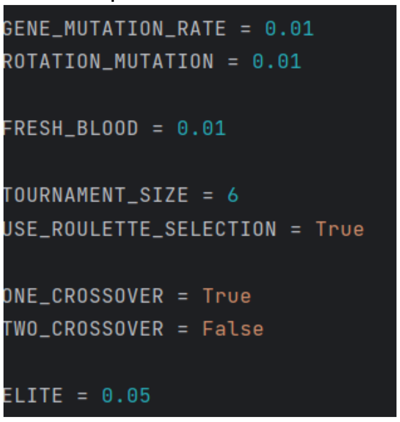


```
11 11  9  6 12 12  6  4 10 10  3  1
11 11  9  6 12  K  6  4 10 10  3  1
 9  K  9  6 12 12  6  4 10 10  3  1
 9  9  9  6  K 12  6  4 10 10  3  1
 9  9  K  6  6  6  6  4 10 10  3  1
 5  5  5  5  5  5  5  4 10 10  3  1
 5  5  5  5  5  5  5  4 10 10  3  1
 4  4  4  4  4  4  4  4  K  K  3  1
 3  3  3  3  3  3  3  3  3  3  3  1
 2  2  2  2  2  2  2  2  2  2  2  1
 7  7  7  7  7  7  7  8  8  8  2  1
```

Example of the output

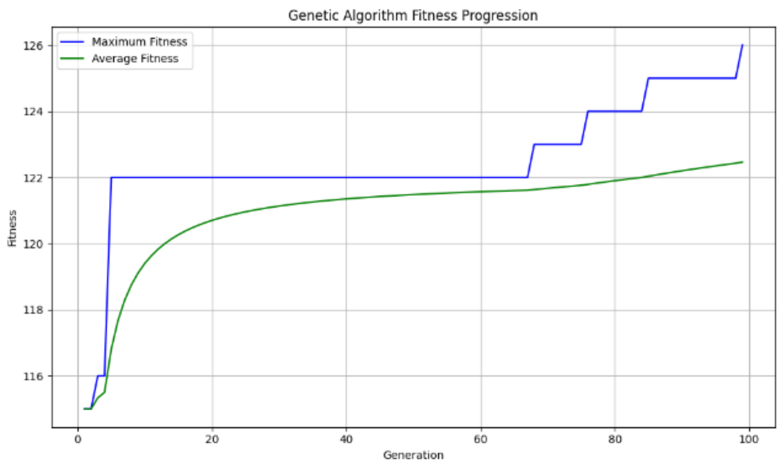
1. Test 1

   The program running using roulette selection and one point crossover
   Current parameters:

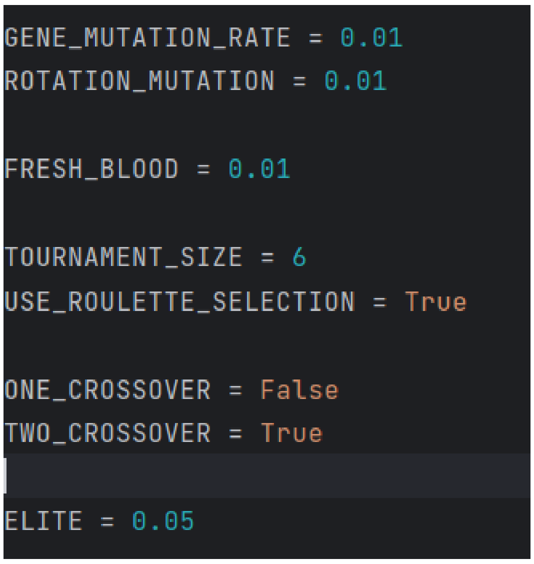


```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = True
TWO_CROSSOVER = False

ELITE = 0.05
```

   result:
   The result is being found in average in 240 generations



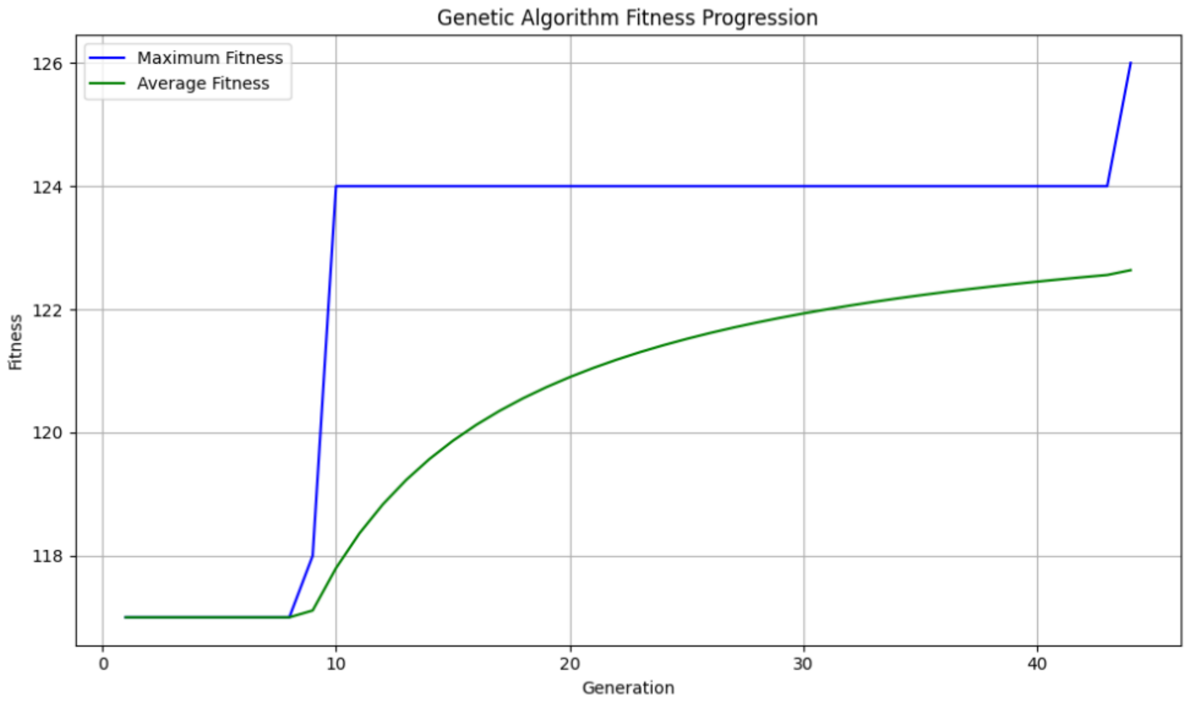
2. Test 2
   The program running using roulette selection and two point crossover
   Current parameters:



```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.05
```

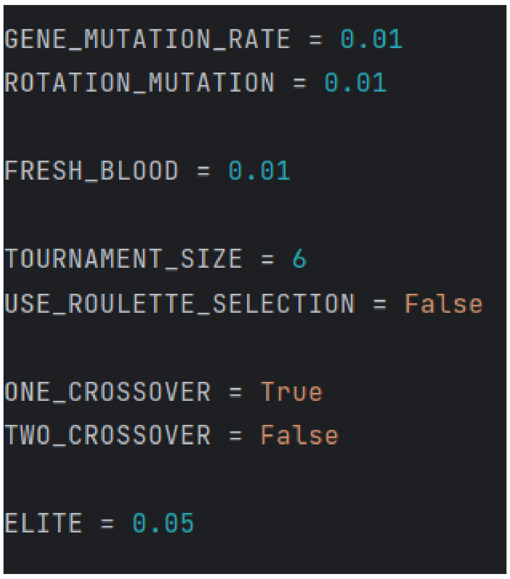
Results:

The result is being found in average in 338,6 generations



3. Test 3

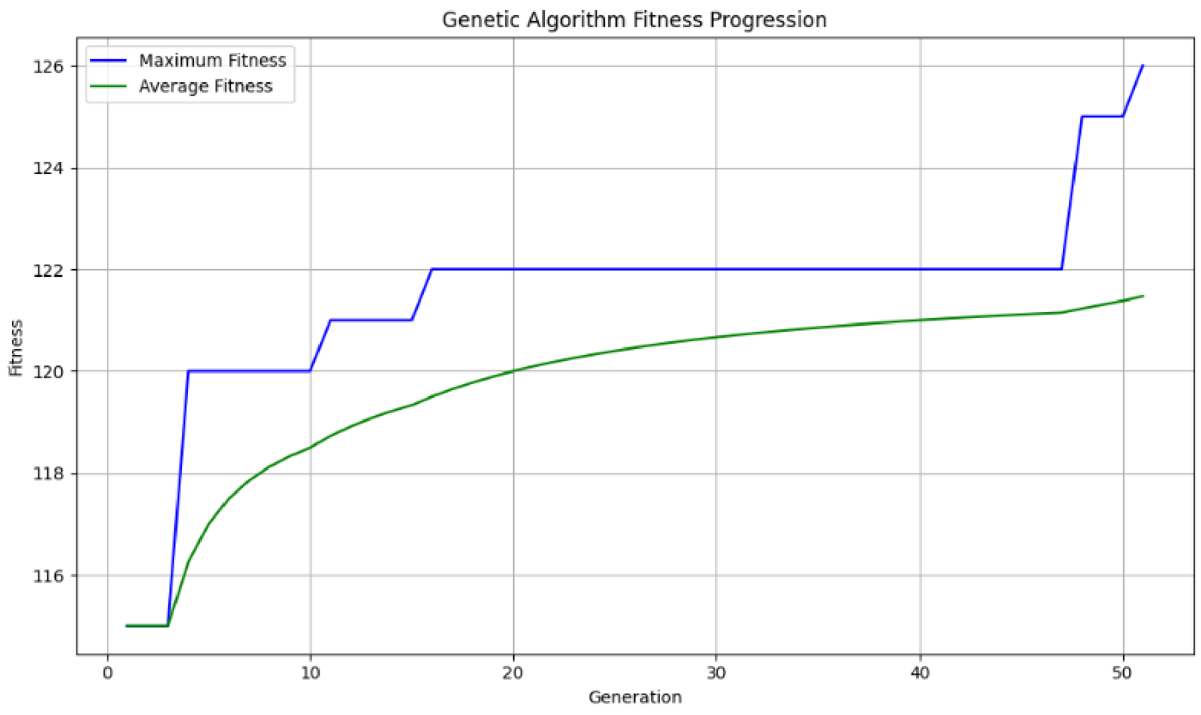
   The program running using tournament selection and one point crossover parameters:



```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = False

ONE_CROSSOVER = True
TWO_CROSSOVER = False

ELITE = 0.05
```

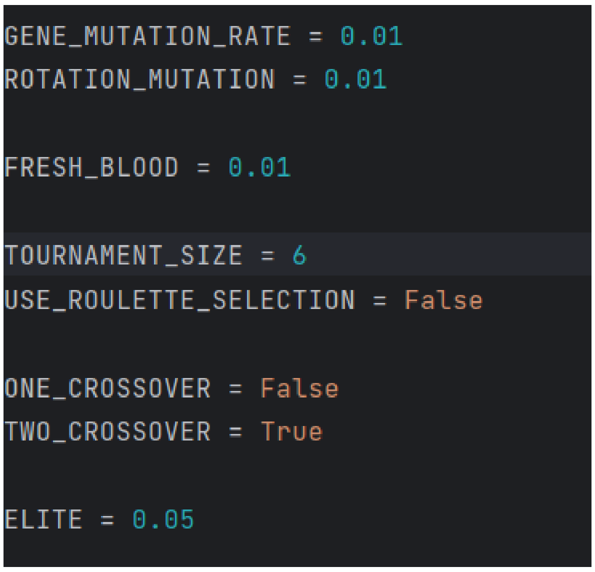
   result:

   The result is being found in average in 487,4 generations



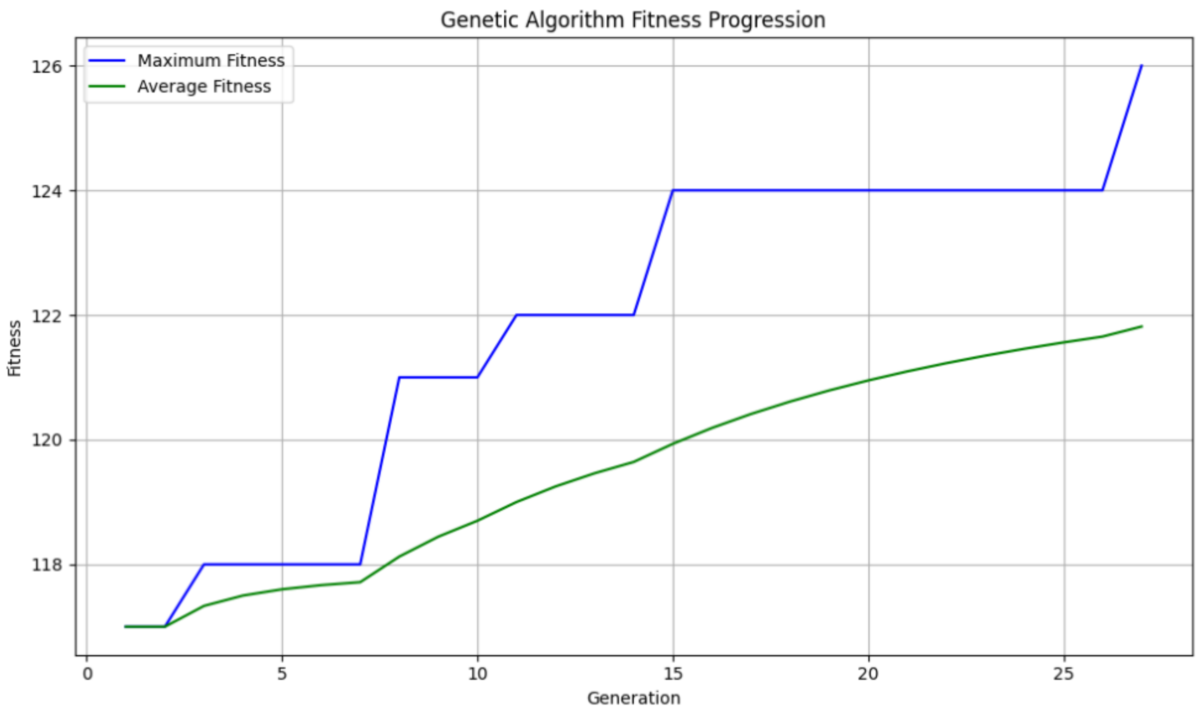
4. Test 4

   The program running using tournament selection and two point crossover parameters:

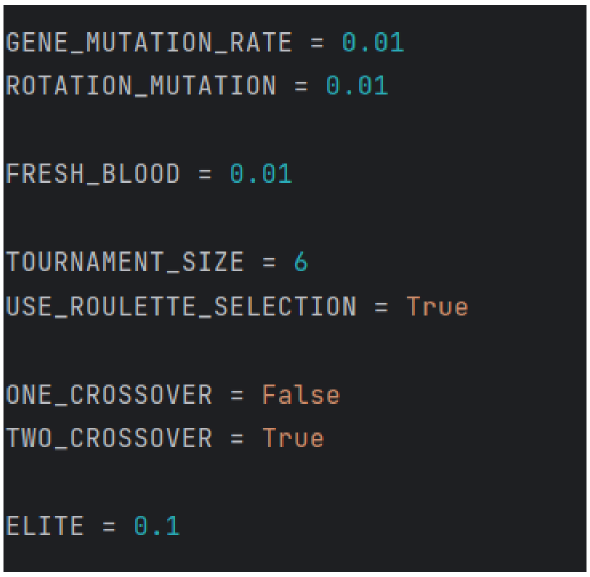


```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = False

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.05
```

result:
The result is being found in average in 494,2 generations

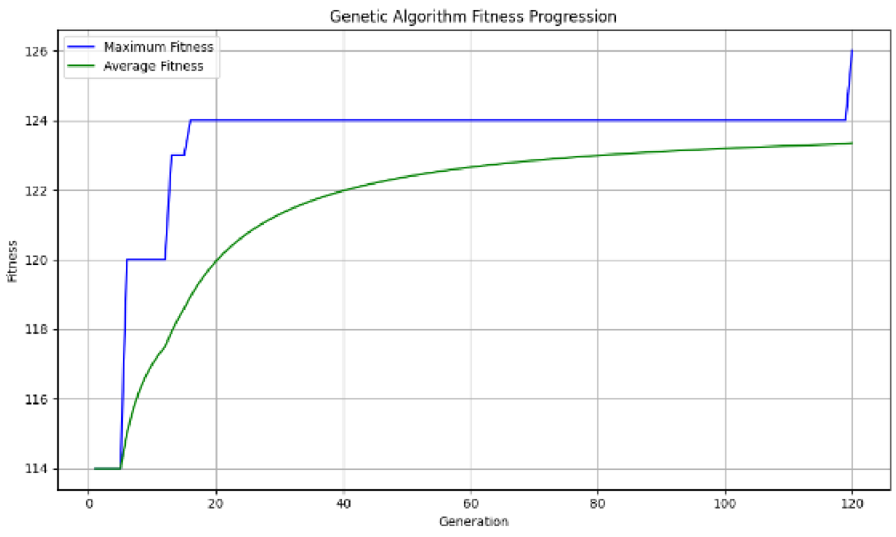


5. Increasing elite percent to 10%
parameters:

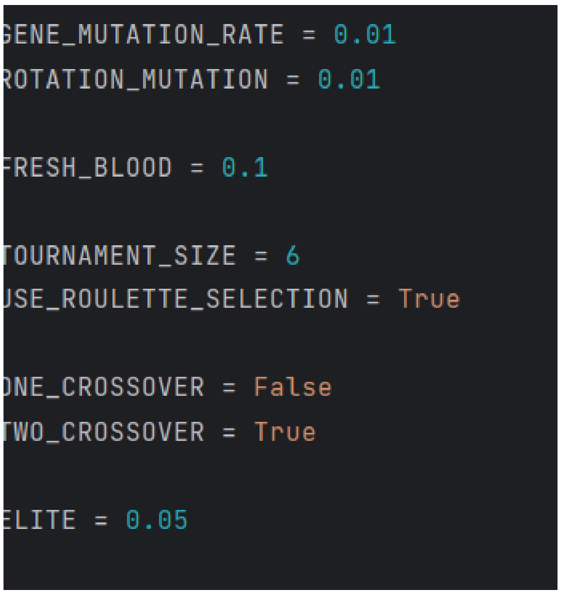


```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.01

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.1
```

result:
The result is being found in average in 223,9 generations

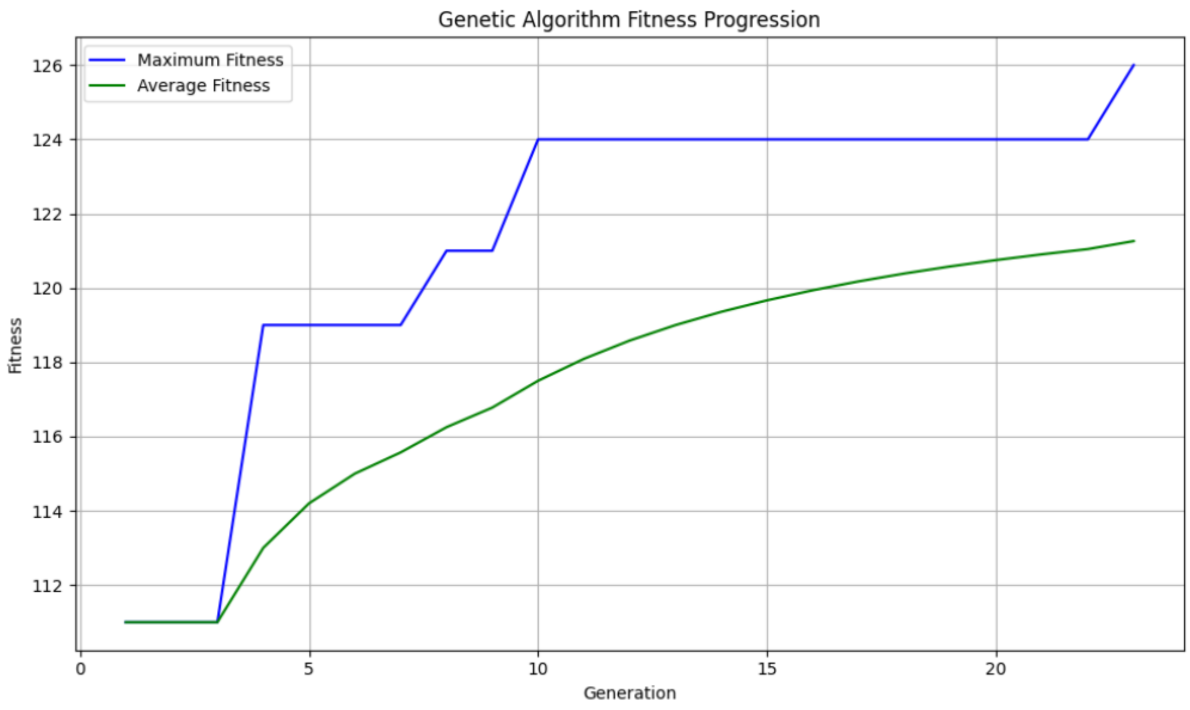


6. Increasing fresh blood percent to 10%
parameters:

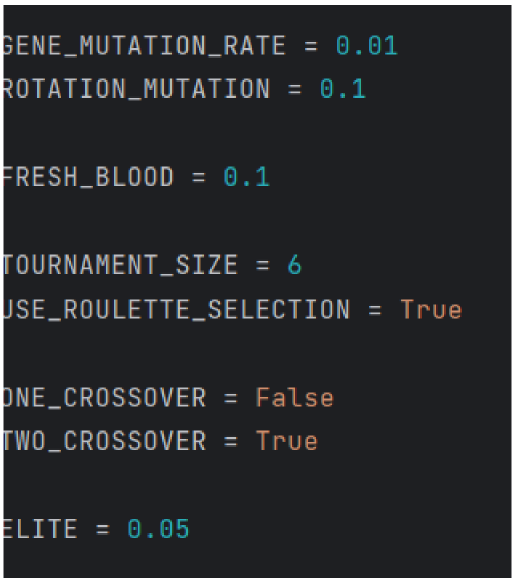


```
ENE_MUTATION_RATE = 0.01
OTATION_MUTATION = 0.01

RESH_BLOOD = 0.1

OURNAMENT_SIZE = 6
SE_ROULETTE_SELECTION = True

NE_CROSSOVER = False
WO_CROSSOVER = True

LITE = 0.05
```

result:
The result is being found in average in 221,3 generations



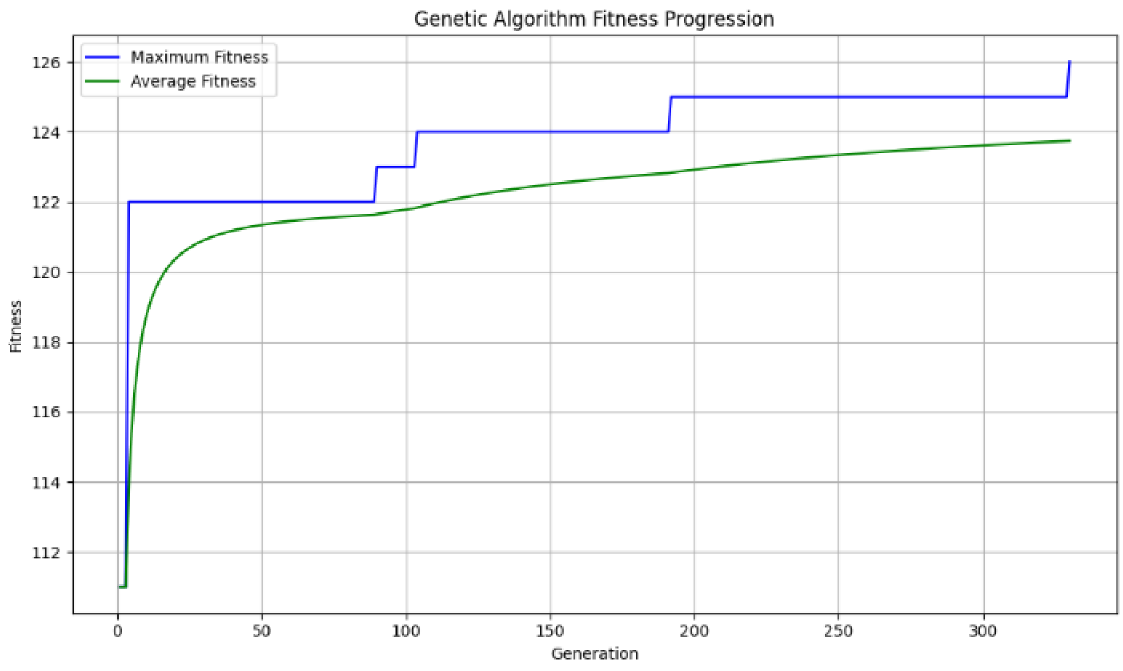
7. Increasing rotation mutation level to 10 %
parameters:



```
GENE_MUTATION_RATE = 0.01
ROTATION_MUTATION = 0.1

FRESH_BLOOD = 0.1

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.05
```

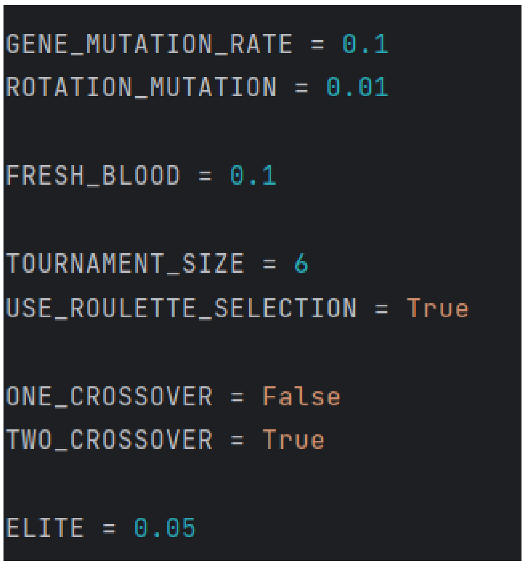
The result is being found in average in 431,3 generations

results:



8. Increasing level of gene mutations to 10 %

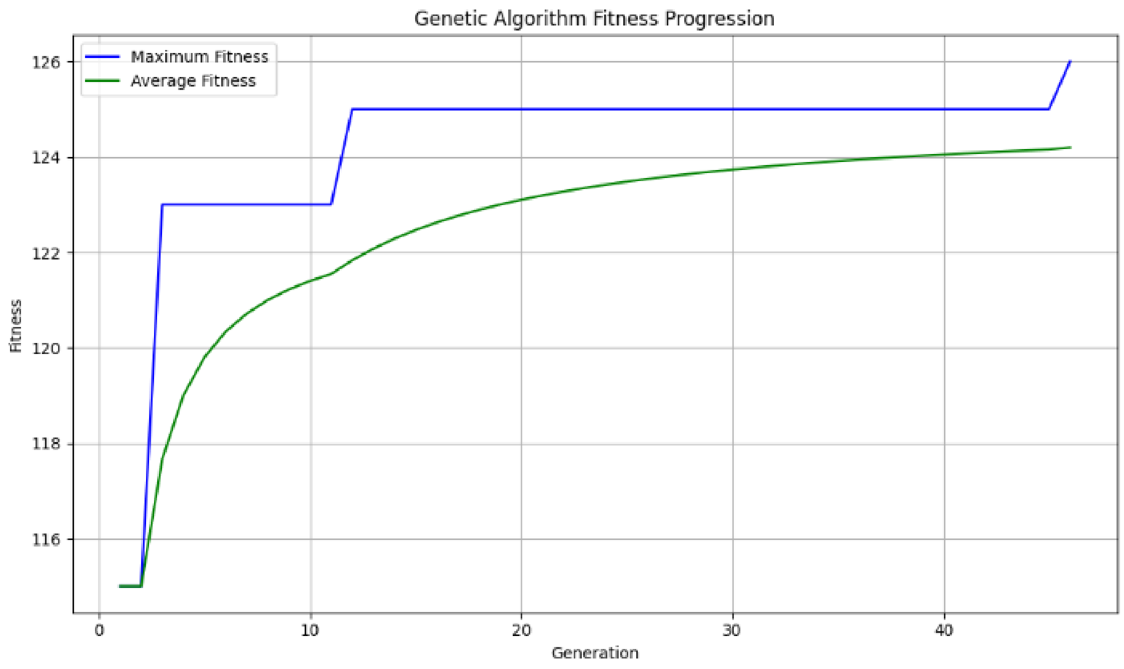
parameters:



```
GENE_MUTATION_RATE = 0.1
ROTATION_MUTATION = 0.01

FRESH_BLOOD = 0.1

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.05
```

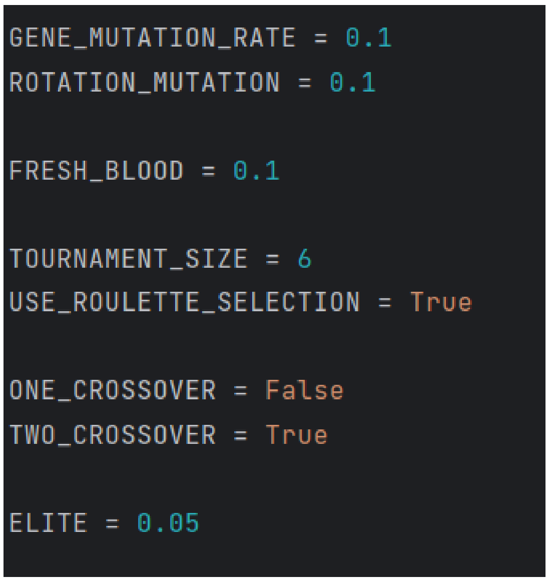
The result is being found in average in 198,9 generations

results:



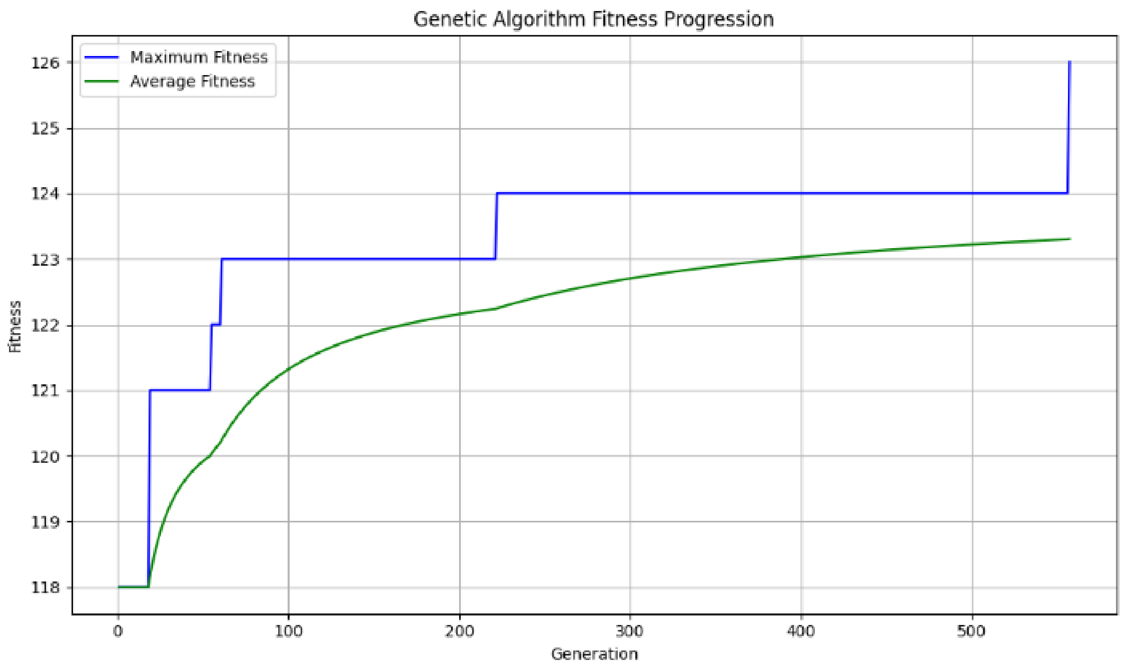
9. Increasing both mutations to 10 % each

parameters:



```
GENE_MUTATION_RATE = 0.1
ROTATION_MUTATION = 0.1

FRESH_BLOOD = 0.1

TOURNAMENT_SIZE = 6
USE_ROULETTE_SELECTION = True

ONE_CROSSOVER = False
TWO_CROSSOVER = True

ELITE = 0.05
```

The result is being found in average in 570,4 generations results:



Conclusion

In conclusion, the Garden Puzzle Solver employs a genetic algorithm to tackle the challenging problem of arranging garden plots within a grid while avoiding obstacles. Through a series of genetic operations, including selection, crossover, and mutation, Monk individuals, each characterized by a set of genes, are continuously evolved over multiple generations. The primary goal is to maximize the coverage of empty cells in the garden while avoiding rocks. The implementation uses a combination of elitism, fresh blood introduction, and a variety of genetic manipulation techniques to explore different combinations of genes and to discover optimal solutions. This approach showcases the power of genetic algorithms in solving complex spatial optimization problems and demonstrates the potential for evolutionary techniques to find efficient solutions in diverse real-world scenarios.