Name: Mouhamadou Barry

EID: 40117811

**Choice of Programming Language**

The Programming language that I choose to solve the Travelling Salesman Problem is Java. Even though my friends have suggested Javascript or Python because of its extensive library, I choose Java because of my interest of the language and the challenge it presented. I have little knowledge of Java as it’s a language that I started learning last year. Because of my lack of knowledge in Java, I faced many difficulties solving the project as a whole. I managed to only do the first two extension. However, in the midst of it all, I’ve learned a considerable amount of Java and its data structure. Overall it was exciting to apply what I already knew and learn new concepts to solve the Travelling Salesman Problem.

**Brief Overview of Genetic Algorithms**

**What is Genetic Algorithms?**

Genetic Algorithm is a heuristic search method used in artificial intelligence and computing. It is used for finding optimized solutions to search problems based on the theory of natural selection and evolutionary biology. Genetic algorithms are made to search through large and complex data sets. Examples of this are facial recognition and fingerprint databases.

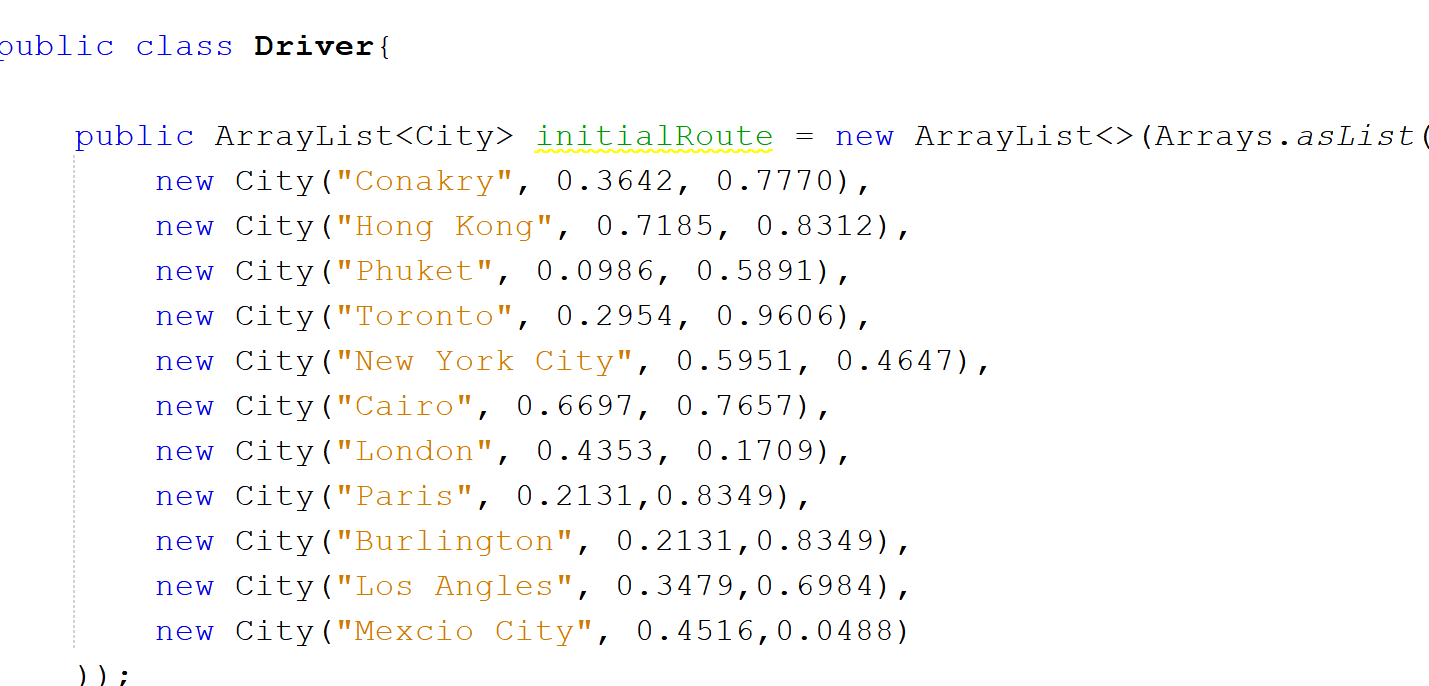
**The Outline of Genetic Algorithm and How Does it Works?**

To create a Genetic Algorithm, you generally have to worry about

1. Creating a random initial population
2. Creating and adding new materials into the initial populations
3. Keeping track of the each individual fitness score
4. Creating a stopping condition

**Initial Population**

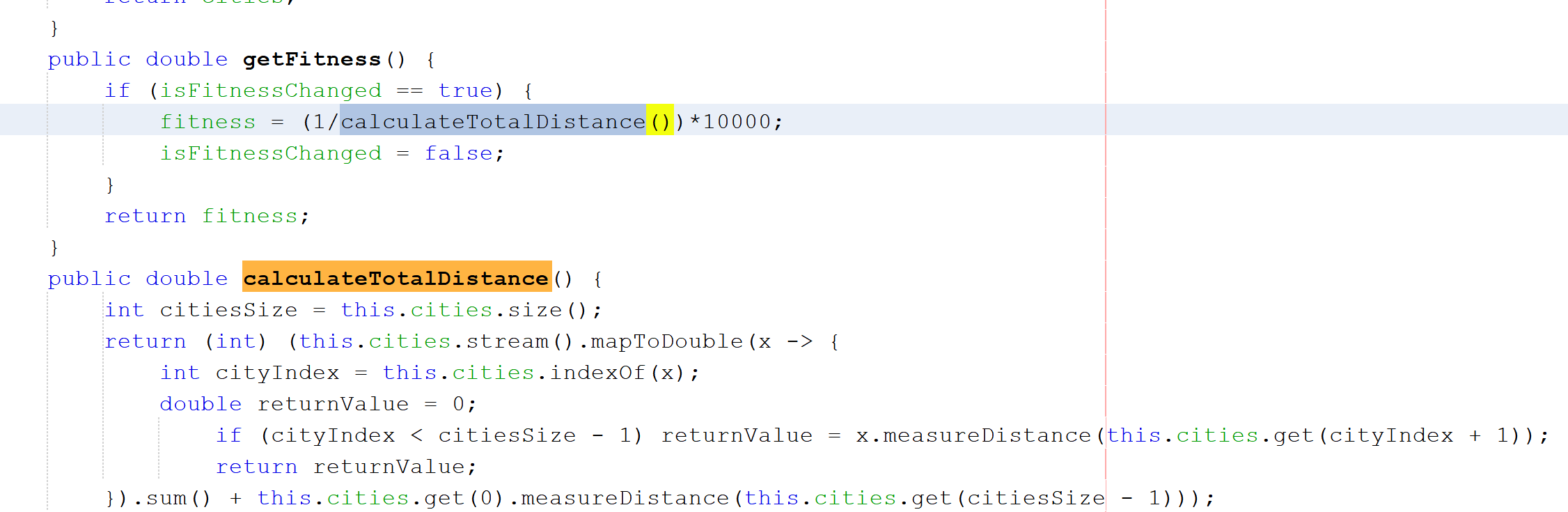
The image below is an example of setting up an initial population for the Travelling Salesman problem



As you can see, the initialRoute is an arrayList that hold all of the 10 X and Y coordinates given to us by the instructions. Please note that instead of just putting coordinates into the population, I’ve also given them abutary city names so I can easily keep track of the points. All of the initial population in the above image has coordinates that ranging from 0 to 1.

**Generating Fitness Scores and a New Population**

Setting up a fitness system or score is an essential part of a Genetic Algorithm. The purpose of a fitness score is to give individuals rating. These rating determine if an two individuals are a superior choice to breed children or not.



In the project code, the individual fitness score is defined by calculating the distance of the city and dividing 1 to it. Afterword it is multiplied by 10000. A fitness score can be calculated and defined in anyway you want it to be.

**Population**

At each step, the genetic algorithm uses the current population to create more children that make up the next generation. The algorithm selects a group of individuals in the current population, called parents, who contribute their genes to their children. The algorithm is usually set up to select individuals that have better fitness values as parents. A Genetic Algorithm can create three types of children for the next generation. The options are Elite children, Crossover children, and Mutation children.

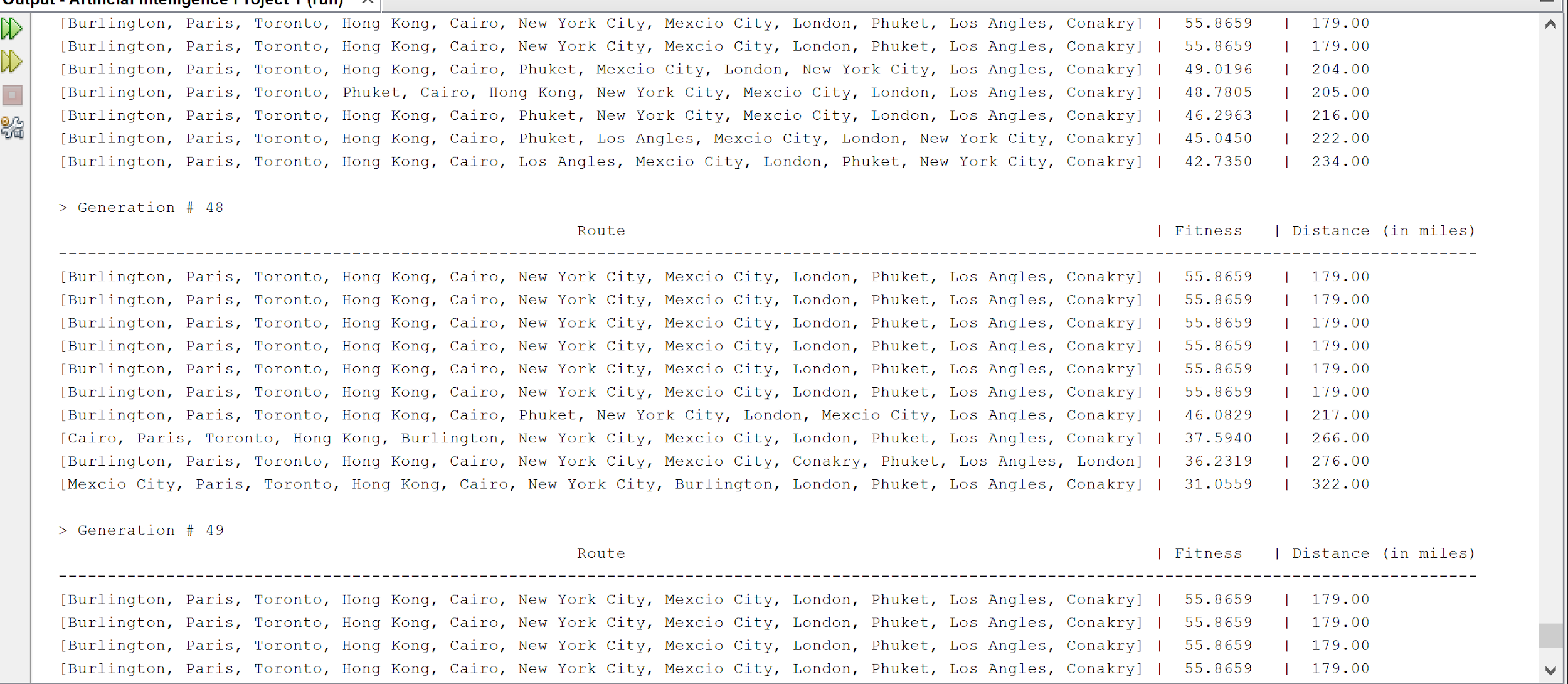
1. **Elite children** are individuals in the current generation with the best fitness values. As a result, these individuals automatically survive to the next generation.
2. **Crossover children** are created by combining the vectors (in my case an ArrayList) of a pair of parents.
3. **Mutation Children** are created by introducing random changes, or mutations, to a single parent.

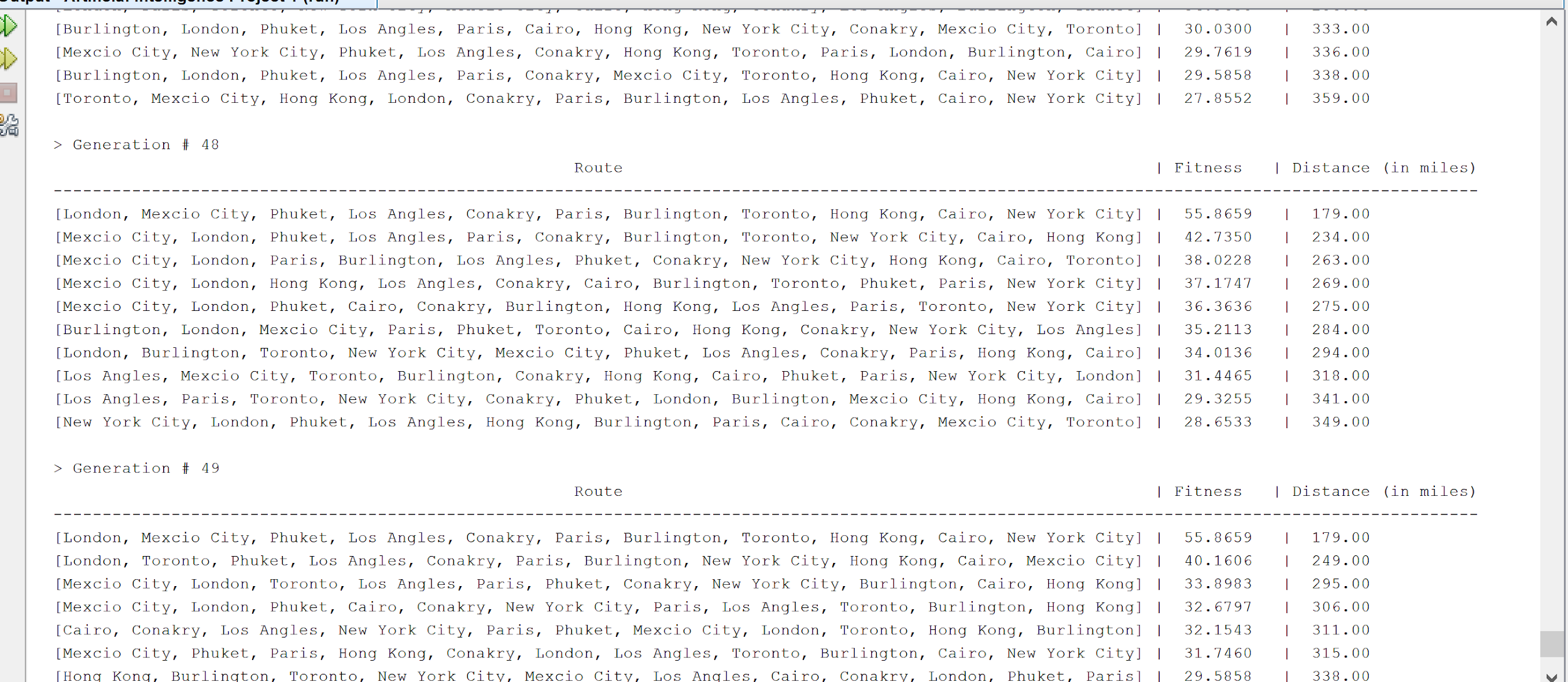
**Mutation and Crossover**

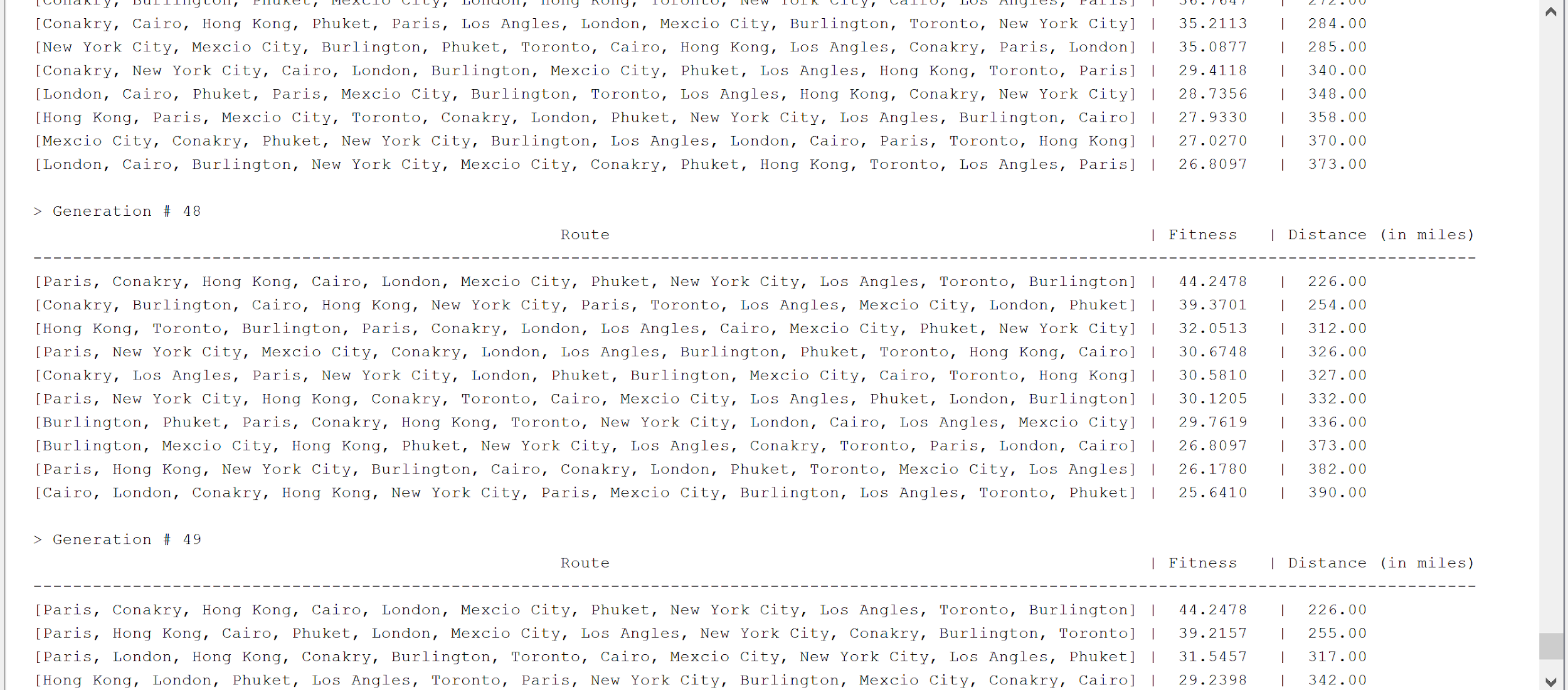
Genetic algorithm creates crossover children by combining pairs of parents in the current population. This enables the algorithm to extract the best genes from different individuals and recombine them into potentially superior children. The selection point of the crossover from the two parent can be defined arbitrary. In my case, I coded the algorithm to always crossover two parents on the 3rd index.

Mutations is needed to change the gene of a parent. Mutation adds to the diversity of a population and increases the likelihood that the algorithm will generate individuals with better fitness values. Something I’ve noticed when I was playing around with the mutation rate is that if I had it too high than It would take more time or generations to get the best distance between cities. What I found interesting is that it was the same case if I had the mutation rate too low. What I took away from this is that is important to set the mutation rate at a reasonable percentage. This is because if your mutation rate is too low, than the population doesn’t change much and starts to lack diversity which can prevent from getting the best distance to begin with. If the mutation rate is too high, than ever generation will have a completely different set of population which will take the search much longer ro run. Down below you will find an example of crossover and mutation from the project code.

This is the result when the mutation rate is 3%



This is the mutation rate at 25%

This is the results when the mutation rate is 90%

**Stopping Conditions for Genetic Algorithms**

A condition can be determine many factors. Some of these factors are

1. Generation - The algorithm will stop when the number of generations reaches the goal you set for it. In this project, Generation was a stopping for my code
2. Fitness limit - The algorithm stops when an individual has a fitness value you are looking for
3. Time limit - the algorithm stops after running a certain amount of time.

The purpose of a stopping condition is to help the search to be more efficient and save time.