## 1. ASSIGNMENT OF MEMBERS

### 1.1 ASSIGNMENT

|  |  |  |  |
| --- | --- | --- | --- |
| Lê Trần Thắng 20204926 | Nguyễn Phúc Thành  20200594 | Hoàng Văn Thắng  20204892 | Lê Thành Thắng  20194451 |

Use case diagram: Le Thanh Thang

General class diagram: Hoang Van Thang

UI: Le Thanh Thang, Hoang Van Thang

Model: Le Tran Thang, Nguyen Phuc Thanh

Controller, UX: Le Tran Thang, Nguyen Phuc Thanh

Report: Le Tran Thang, Le Thanh Thang

Slide: Hoang Van Thang

### 1.2 CARIFY

To complete the application, we refers to the algorithm on [Traveling Salesman Problem with Genetic Algorithms in Java (stackabuse.com)](https://stackabuse.com/traveling-salesman-problem-with-genetic-algorithms-in-java/) and the GUI based on [Traveling Salesman Problem with Genetic Algorithms in Java (stackabuse.com)](https://stackabuse.com/traveling-salesman-problem-with-genetic-algorithms-in-java/)

## 2. MINI-PROJECT DESCRIPTION

### 2.1 PROJECT OVERVIEW

The mission of our project is to build an application which visualizes the process of solving the TSP (travelling salesman problem) by the genetic algorithms.

Due to the main purpose of visualization is to help user get a better insight about how an algorithm works, we have put some restrictions in our application:

1. The number of cities must be in range from 1 to 500.

2. The number of generations of the algorithm must be positive (>0).

### 2.2 DESIGN REQUIREMENTS

- On the main menu: title of the application, 3 buttons for genetic algorithms for user to help, solve the TSP and quit.

+) Help menu: show the basic usage and aim of the program.

+) Quit: exits the program. The application should ask for confirmation before closing.

- In the demonstration:

+) A button for starting the simulation, namely, “Start”.

+) A visualization for the individual that currently holds the best fitness, in array-structure.

+) A visualization for the individual that currently holds the best fitness, in graph-structure.

- For a better understanding and visualization, we have added two more feature in our application:

+) Reset, pause, continue, and load button: instead of letting the application automatically run the visualization user can manually controls the process to see clearly what happened.

2.3 USE CASE DIAGRAM EXPLAINATION Diagram

Description automatically generated**Figure 1.** Use Case Diagram

Based on all the requirements we decided to develop 3 use cases (as shown in the figure of our application).

To be more specific:

1. Solve TSP

* The Solve TSP use case will take all user command from the GUI such as: the number of cities in the problem, the number of generations of the genetic algorithm.
* Also, the Solve TSP use case will notify the user if there is any problem with the input.
* Run the algorithm and visualize it to the user on the GUI.
* Wait for the command of the user whether he/she wants to control the process of the algorithm.

1. Show instruction use case will get user to help menu for more details about program and how to use it.
2. Exit program use case will get user confirmation to close the program.

## 3. OOP DESIGN

### 3.1 GENERAL DESIGN

Diagram

Description automatically generated

**Figure 2.** General Class Diagram

Packages:

+ Package “model”: Stores all basic data structures and objects to implement the genetics algorithm on traveling salesman problem.

+ Package “screen”: Stores the main screen class, dialog class, drawPane class, and controllers to visualization in GUI.

### 3.2 SPECIFIC DESIGN

#### 3.2.1 Model package

Diagram

Description automatically generated

**Figure 3.** Model Package Diagram

**Point:**

+ Attributes:

* int x, int y: The location of point in the TSP.

+ Method

* equals(): Checks if the two points have the same location.
* distance(): Calculates the distance between two points.
* Getters and setters methods: return the location of Point

**TSP:**

+ Attributes:

* int n: The number of cities.
* float[][] distance: the distance between every pair of cities in TSP.
* ArrayList<Point> points: contains the location of all cities

+ Constructor

* TSP(int n): Create the problem with n cities.

+ Method

* generatePoint(int n): Generates n Points as n cities with our defined restrictions.
* calculate(): Calculates the distance every pair of cities in TSP.
* Getters and setters methods: returns the travelling salesman problem.

**Chromosome extends ArrayList<Integer>:** Our defined object to implement the genetic algorithm

+ Method

* toString(): transform an ArrayList to a String as the output of the route.

**GeneticAlgorithm**:

+ Attributes:

* int generateSize = 100: the number of chromosomes to keep in a generation.
* ArrayList<Chromosome>: the population keep after a generation.
* Int step = 0: the current step of the algorithm.

+ Constructor

* GeneticAlgorithm(int n): Create the genetic algorithm with n chromosomes.

+ Method

* generateChromosomes(int n): Generates n Chromosomes.
* crossover(Chromosome parent1, Chromosome parent2): Creates children chromosomes based on the 2 parent chromosomes.
* mutate(Chromosome child): Swap two genes of the chromosomes with fixed rate.
* select(): Keeps the best chromosomes to create more children for the next generation.
* objectiveFunction(Chromosome c): Abstract.
* solveNext(): Implements the genetic algorithm by one generation.
* Getters and setters methods: returns the genetic algorithm.

**GAForTSP:**

+ Constructor

* GAForTSP(int n): Create the genetic algorithm solving traveling salesman problem of n cities with n chromosomes .

+ Method

* objectiveFunction(Chromosome c): Calculates the fitness of chromosome in traveling salesman problem.

#### 3.2.2 Screen package

Diagram

Description automatically generated

**Figure 4.** Screen Package Diagram

**MessageDialog:**

+ Constructor

* MessageDialog(): Creates the JDialog to informs about mismatch inputs, …

**QuitDialog:**

+ Constructor

* QuitDialog(): Creates the JDialog to get confirmation to quit the program.

**HelpScreen:**

+ Constructor

* HelpScreen(): Creates a JFrame to shows the basic usage and aim of the program.

**DrawPane:**

+ Attributes:

* Chromosome route: the route that the chromosome represents.
* Chromosome minRoute: the route that the best chromosome represents.
* float pointSize = 3.5: the size of point in the GUI

+ Constructor

* DrawPane(): Creates a Pane visualizes the algorithm.

+ Method

* paintComponent(Graphics g): draw the route and cities on the pane.
* Getters and setters methods: set the route and cities to draw the pane.

**MainScreen:**

+ Constructor

* MainScreen(): Creates a main screen of the program.

### 3.3 DESIGN EXPLAINATION

Our GUI is design based on the MVC model, the model package will contain all the code about the algorithms to manages the data, logic, and rules of the program. The screen package is written by Swing will contains both the code about controller and view. The controller defines the interactions between the model and view, it will accept input and converts it to model or view. The view will represent the information to user.

In model package, we have class TSP aggregates from class Point to initialize the TSP and GAForTSP class extends from GeneticAlgorithm class and aggregates from TSP class to make genetic algorithm for solving TSP. Also, we have class GeneticAlgorithm aggregates from class Chromosome to implements genetic algorithm.

The Screen package contains screen class and controllers to take input, control the model and represent it on GUI for the user.

One feature about our program is that we run the algorithm and stimulate the UI simultaneously as we assigned it to the timer which will run with a given period of time after the user clicks the start button.