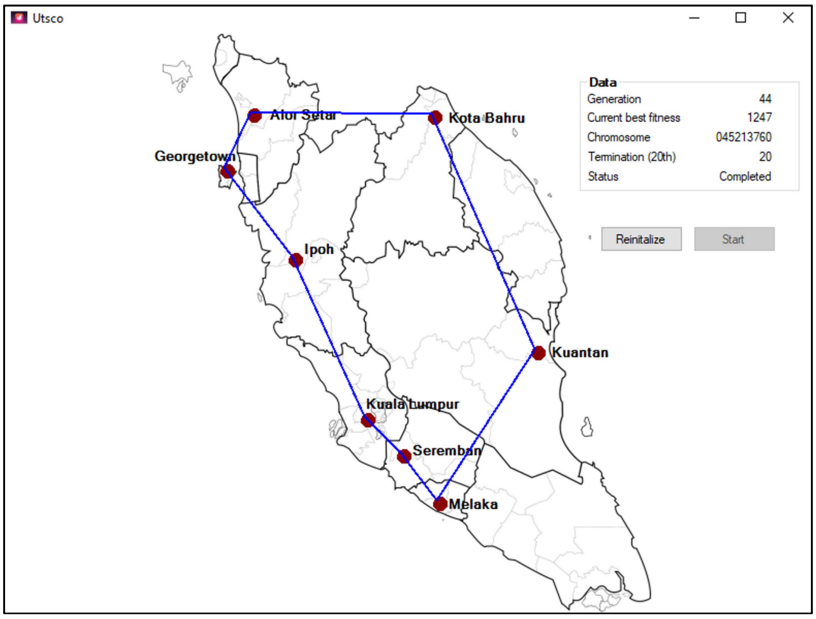
Problem definition

The problem that I am solving with this project is The Travelling Salesman Problem. The problem was first formulated in 1930 and has been one of the most intensively studied problems in optimization. The application involved finding shortest distance for a textbook salesman in planning his visit to potential clients in all eight major cities in Malaysia. The textbook salesman sets foot to travel between eight cities without visiting a city more than once starting from Kuala Lumpur and back to Kuala Lumpur. The search space of this problem involves every possible permutation of routes that visit each city once. The total possible routes are 8! = 40320 is impossible to find the path manually, hence genetic algorithm is used as genetic algorithm is great for finding solutions to complex search problems.

The input parameters that are given to solve the problem are the distance between cities. I am able to compute and store the data given into 2-Dimensional integer array [8][8] and put it into an Environment class. From there, I design an algorithm to calculate the total distance requires for travelling each tour.

The output parameters will produce the best Tour chromosome and its fitness. Below illustrates how the salesman travel from KL back to KL using lines and maps. Likewise, best fitness and best tour of the current generation is shown. The figure below showns an example of all the output parameters



Encoding and initialization procedure

Intended for encoding, permutation encoding is used as it prevents repetition of items and represents a sequence of items. Permutation encoding is a useful solution for problems that have specific order like Travelling Salesman Problem. The chromosome will describe order of cities, in which salesman visit them.

It is clear that gene is represented by “city” and chromosome represented by “tour”. The idea is that each integer array represents each city. For example, [Kuala Lumpur] = [0], [Kota Bahru] = [1] and so on. Hence, the city object contains its city_ID which represent the described data. In this regards, a chromosome of selected cities will be produced.

Upon initialization, each individual creates a permutation featuring an integer representation of a route between eight cities from Kuala Lumpur back to Kuala Lumpur with no repetition features. I use a non-repeating random generator to produce the chromosome. A corresponding array with the string equivalent of these indexes is created to output when a solution is found.

[0, 4, 1, 2, 6, 5, 7, 3, 0] = [“Kuala Lumpur” , “Seremban”, “Kota Bahru” , “Kuantan” , “Ipoh” , “Melaka”, “Georgetown”, “Alor Setar”, “Kuala Lumpur”]

Subsequently, we add the following chromosomes into a list of population, hence Population.p[0] is chromosome 1.The population size is set to 100 but it could be easily changed by accessing the “Environment class”. The following is provided for illustration and clarification purpose.

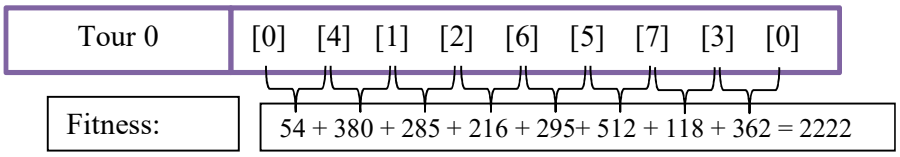
Population

Tour 0	[0] [4] [1] [2] [6] [5] [7] [3] [0]
Tour 1	[0] [4] [5] [2] [1] [3] [7] [6] [0]
Tour 2	[0] [6] [7] [3] [1] [2] [6] [4] [0]

Note: The gene of each chromosome always start from Kuala Lumpur and return to Kuala Lumpur. Hence, it starts from [0] and end with [0]

Selection operator

There are many type of method when performing selection in genetic algorithm such as roulette wheel selection and tournament selection. However, the method that we are using is the fitness-based selection. In each generation, the fitness of every chromosome in the population is evaluated and sorted from the best to weakest. The fitness is valued according to distance travels by the salesman. Hence, the lower the distance travelled, the lower the fitness which means better fitness score. The following image illustrates how fitness is obtained.



As the goal is to find the optimal minimum fitness score with lowest distance travelled, the following is the selection operation steps:

1. Evaluate and sort the population from best to weakest (lowest distance travelled to furthest distance travelled).
2. Perform Elitism that keeps 10% of fittest population to the next generation.
3. Then, the remaining 90% of the population is produced by fitness-based parent selection.
4. Parent selection uses 50% population of the best tour, selects it randomly as parents to mate and produced better offspring.
5. Note: We also created a function that replaces chromosome duplication by checking fitness; this idea is to prevent **Premature Convergence** which is an undesirable condition for Genetic Algorithm.

Crossover operator

As the solution requires that no city to be visited more than once, using a classical crossover operator may often lead to generating weaker offspring. A conventional crossover operator may select one half to copy from the first section and the remaining half to copy from the second. This does not prevent copying duplicate cities into the offspring chromosome and will result in a penalty for visiting the same city more than once.

```
//Classical Crossover Operator
[ 0 , 6 , 2 , 7 , | 1 , 4 , 3 , 5 ]      //Parent 1
[ 2 , 1 , 7 , 0 , | 6 , 5 , 4 , 3 ]      //Parent 2

[ 0 , 6 , 2 , 7 , | 6 , 5 , 4 , 3 ]      //Offspring 1
[ 2 , 1 , 7 , 0 , | 1 , 4 , 3 , 5 ]      //Offspring 2
```

Therefore, I use ordered crossover way to help preserve the non-repeating feature of the parent chromosomes.

1. To create the offspring, copy first 5 gene of parent 1 chromosome to the offspring's chromosome.
2. Choose the other 3 valid non-repeating numbers from parent 2 in the parent 2 order and place it to remaining offspring chromosomes.
3. Add gene [0] = Kuala Lumpur to the offspring's chromosome and its final destination is at Kuala Lumpur.
4. However, there are 50% chance of parent 1 gene being more dominant than parent 2 genes and 50% chance of parent 2 gene being more dominant than parent 1 gene.
5. The offspring contain more genes of 5 from the alpha parent than 3 from non-alpha parent.

```
//Ordered Crossover Operator
[ 0 , 6 , 2 , 7 , 1 , 4 , 3 , 5 ]      //Parent 1
[ 0 , 1 , 5 , 2 , 6 , 4 , 7 , 3 ]      //Parent 2

[ 0 , 6 , 2 , 7 , 1 , 5 , 4 , 3 ]      //Offspring
```

Mutation Operator

Since this specific problem is a permutation based encoding, we need to avoid creating repetition of cities visited. The conventional mutation method of replacing a value with a random city will result in duplication and does not fit the problem. Hence, Swap Mutation Operator is used as it prevents repetition but changes the order of the Chromosome/Tour.

Every offspring that is created from mating will have a mutation rate of 0.1 which is 10% to have its gene mutated.

```
[0, 6, 2, 7, 1, 4, 3, 5 , 0] --> [0, 4, 2, 7, 1, 6, 3, 5, 0]
```

Termination Criteria

The termination criteria that we set are simple to understand. If the fitness is the same for 20 consecutive generations, the algorithm will stop and terminate. To put it simply, we terminate when highest ranking solution's fitness has reached a plateau such that successive iterations no longer produce better result.

Reference

1) Russell, S.J. & Norvig, P. (2009) Artificial Intelligence: a modern approach. (3rd ed.). Prentice Hall.
2) Jacobson, L. (2012, August 20). Applying a genetic algorithm to the traveling salesman problem. Retrieved November 19, 2018, from http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5