Sam Ewert

Homework 2: AI for game  
Part 1: Genetic algorithms (paper and pencil type of problem)  
We have seen the example of Hello World in genetic algorithm (C++ code posted on Canvas). In   
general, a genetic algorithm could be described as follows:  
Generate the initial population  
Compute fitness  
REPEAT  
 Selection  
 Crossover  
 Mutation  
 Compute fitness  
UNTIL population has converged (in the case of Hello World, when the string we generated   
look exactly like “Hello World!”  
Now, assume that you are asked to write a pseudocode of a genetic algorithm to find an optimal   
road for a new Amazon delivery truck driver who is also new in town. He departs from Amazon   
warehouse and has to go through n houses by the end of his day (h1,h2,...,hn). Given that we   
know the distance between these houses (called dij = distance(hi, hj), and dsi = distance(s,hi) with   
s being the Amazon warehouse. At the end of the day, the truck will go back to the warehouse s.   
So a typical road would look like: s -> h2 -> h1 -> h3 ....-> hn-1 -> hn -> S.  
Please use genetic algorithm with the above framework to write a pseudo code to find a road that   
minimize the distance that he needs to travel.  
In your pseudocode, please feel free to suggest the size of the initial population, the fitness   
function, choose how you do cross-over and mutation. You don’t have to implement this pseudo   
code but will be graded on the details you could provide so that people who are implementing it   
would find it easy to work on.

**Pseudo Code Starts Here**

**Genome Class:**

Private Dynamic Array of integers *geneSequence*: stores the road path

Private Integer *fitness*: calculated fitness for the genome

Constructor that accepts a Dynamic Array and assigns it to the corresponding class variable

Getters and Setters for each class variable

**Genetic Algorithm class:**

Final static int *locations*: number of delivery locations

Final static int[][] *map*: stores distances from each location to each other location

Final static int *populationSize*: size of the population (ie 2048)

Final static int *maxIterations*: max times the algorithm will run with no change in best fitness before ending (20)

Static Dynamic Array of Genome objects *genomes*: stores all the genomes

Generate the initial population method

For the population size

Create a dynamic array *gene* that stores one of each location (1 – *locations*)

Randomly shuffle all the values in *gene*

Add zero at the front and end of *gene* (represents starting and ending at the warehouse)

Create Genome object *g* and pass *gene* as an argument

Add *g* to *genomes*

Compute Fitness method

For each Genome in *genomes*

Get the Genome *g*

count = 0

Iterate through the locations in *g*’s geneSequence

Add distance values to *count* using *map* for distance values

Store count in *fitness* for *g*

Repeat until converged (until the fitness has not improved for a definitive number of interations)

Selection method

Sort *genomes* based off *fitness* (lowest fitness to highest)

Top 1% are guaranteed a spot in the next generation (elite)

Mutation method ~~Crossover~~ Crossover is inefficient when each location can only be visited once

For lower 99% of population of *genomes*

Randomly swap location values in the *geneSequence*

Compute Fitness method – Look at previous Compute Fitness method