Genetic Algorithm: Robot Controller in a Maze

Anish Surti, Akash Jagtap

**Problem**

This Genetic Algorithm is designed to solve the "Robot Controller in a Maze" problem. The problem poses the situation: given a maze which is represented as a 2d array of integers and an instruction set, the Robot will attempt to navigate to the finish line.

|------|-------|-------|------|------|-----|

|begin | route | wall | wall | wall | wall|

|------|-------|-------|------|------|-----|

|wall | route | wall | wall | wall | wall|

|------|-------|-------|------|------|-----|

|wall | route | route | wall | wall | wall|

|------|-------|-------|------|------|-----|

|wall | wall | route | wall | wall | wall|

|------|-------|-------|------|------|-----|

|wall | wall | route | wall | wall | wall|

|------|-------|-------|------|------|-----|

|end | route | route | wall | wall | wall|

|------|-------|-------|------|------|-----|

Here, the **route** represent the correct route through the maze, **wall** can't be navigated through. You can follow the routes visually to find the correct path through the maze.

Our aim is to find out a fitness score for a path; it is this score that the genetic algorithm will optimize.

**Implementation Design**

*Genetic code:* Each gene is an integer, either 0 or 1 in an array list.

*Gene expression:* Each pair of genes corresponds to an action, which are represented as below.

00 - do nothing

01 - move forward

10 - turn clockwise

11 - turn counter-clockwise

*Fitness function:* Here, we must evaluate how good the robot is at navigating a maze. The fitness function checks if the co-ordinates of the robot match with the robot route in the matrix and if the position is not visited then the fitness is incremented.

*Sort function:* This function sorts the solutions based on their fitness values (high to low).

*Mutation:* We are checking if the mutation rate is greater than the randomly selected value using Math.random() and then we simply randomly flip 0s to 1s and vice versa.

*Crossing Over:* Two solutions are selected from the population. First, we check if the crossover rate is greater than the randomly selected value using Math.random() then we select a random swap position and depending on where we are with respect to that random swap position, the crossover solution is either replaced by solution 1 or solution 2

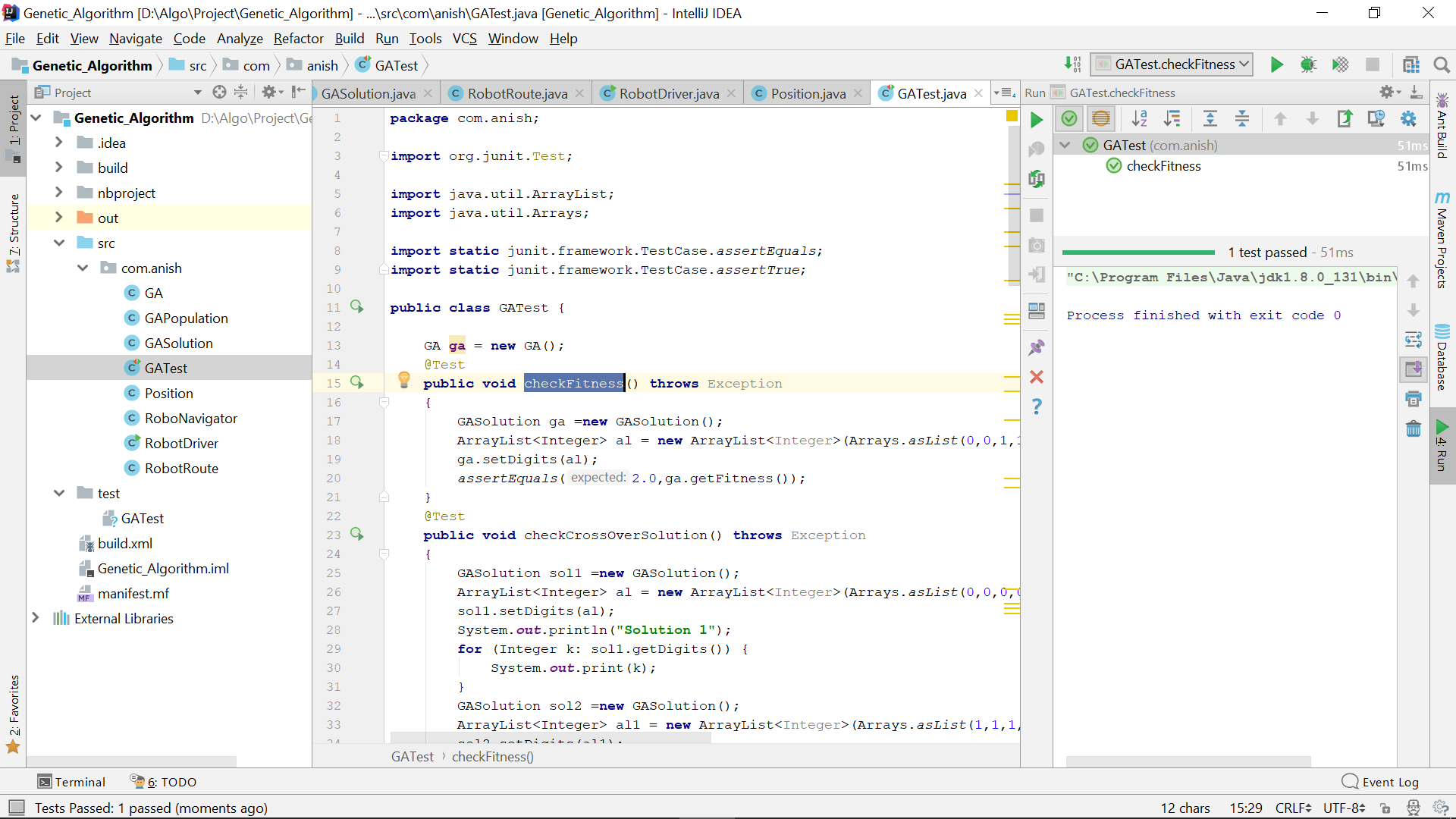
*Tournament Selection:* We have instantiated the tournament population based on tournament selection size which is 4 in our case. The tournament population is randomly populated with 4 random solutions from the past population and then choosing the best of those.

*Evolution:* The population is stored as an array list. The population is seeded with 10 individuals with genes that have been added randomly. We then calculate the fitness function for each individual, perform a sort by fitness method and keeping only the elite individuals (which is 1 in our case) from the population. The remaining individuals then reproduce, doubling the population again.

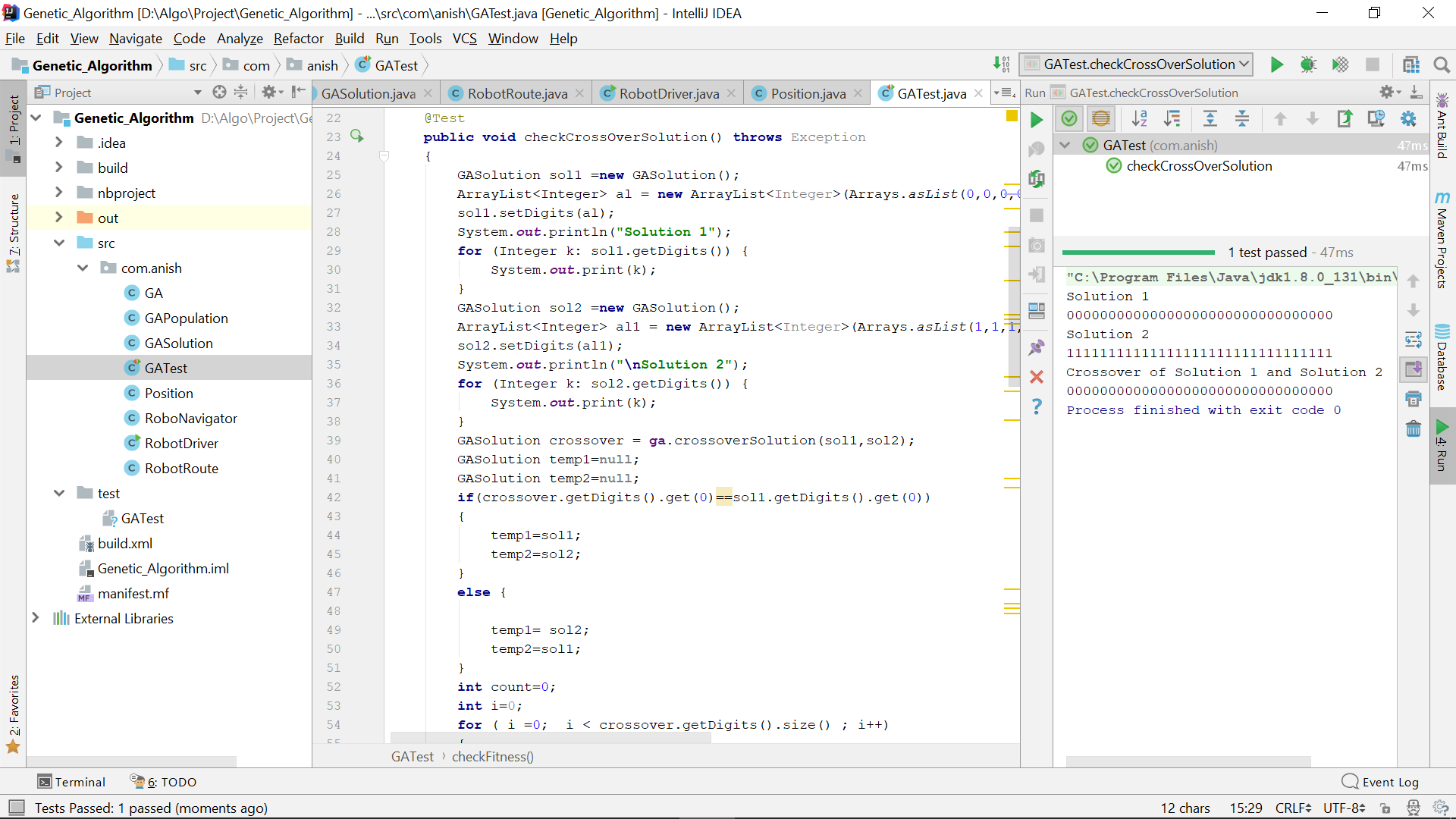
For each generation, the fittest solution is logged along with the highest fitness score. The evolutionary process terminates after 1000.

Unit Tests :-

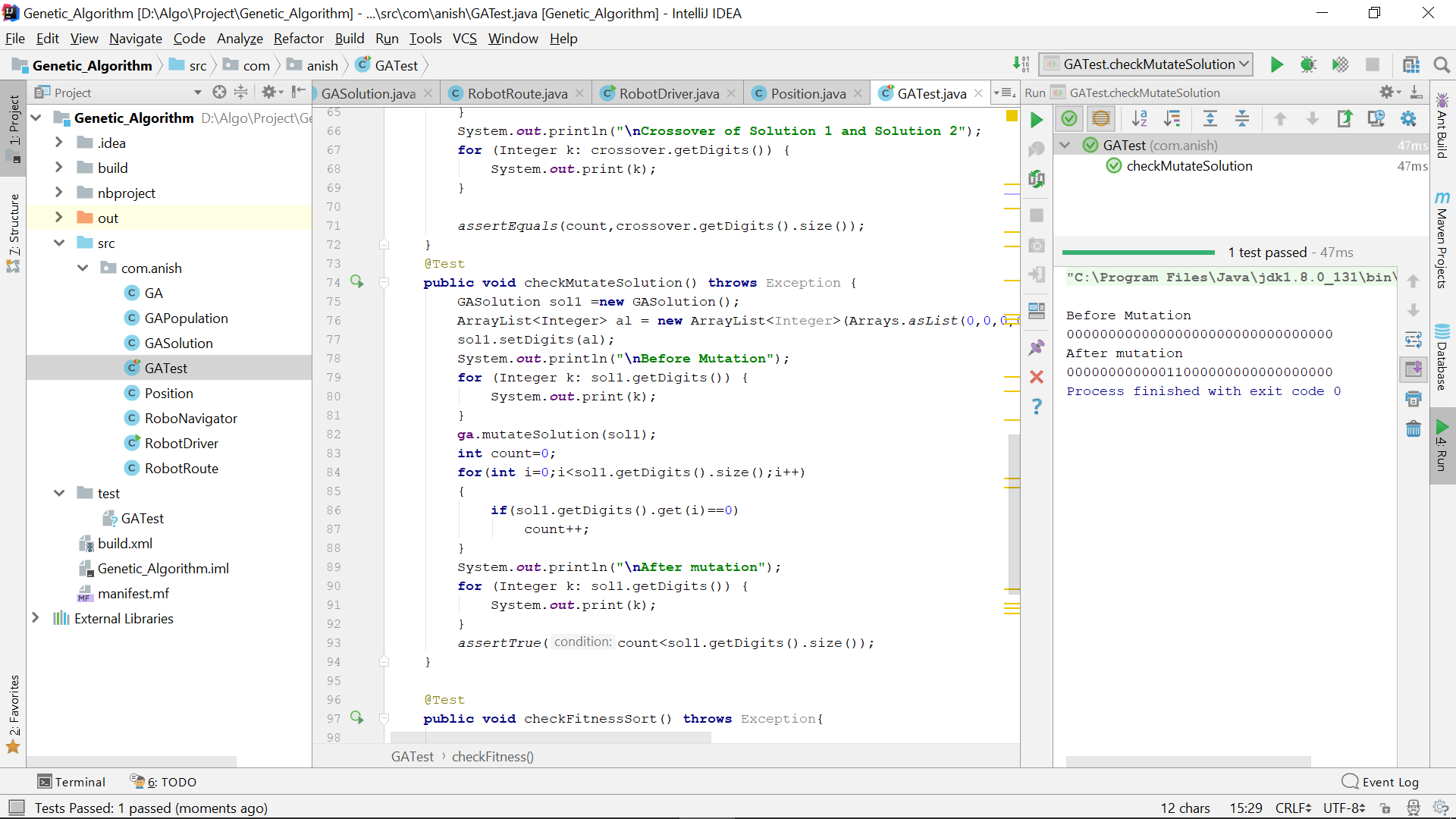
For Fitnessfunction:



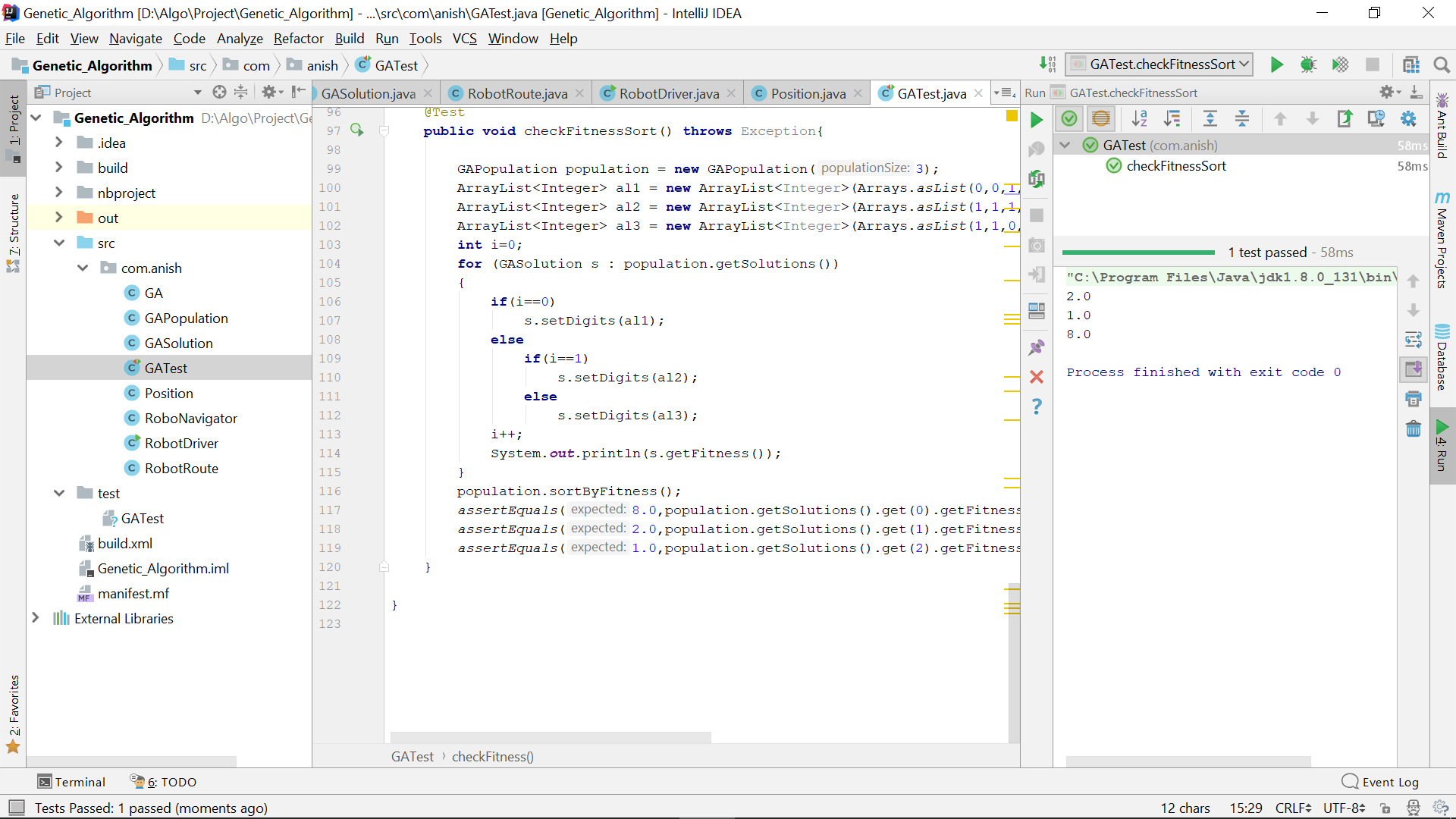
For CrossoverFunction:



For MutationFuntion:



For FitnessSortFunction:



The output of the program is shared in the “**OutputOfTheProgram.docx**” file.