INFO6205 Final Project-Genetic Algorithms

Robot Controller

Group 525

Group Members:

Tse-Hsi Lin

Yu-Chiao Huang

Instructor:

Prof. Robin Hillyard

Implementation

Problem Statement:

The problem we are going to solve is designing a robotic controller that can use the sensors to walk through the maze and arrive the destination without crashing into the wall. The robot has six different direction sensors, three on the front, one on the left, one on the right and one on the back. The robot can take four actions: move one-step forward, turn left, turn right, or stay. Each route can be one solution and can be scored, we can find the fittest solution by comparing the scores we get. The purpose of this project is not to train a robot to solve the maze, our purpose is to automatically program a robot controller with six sensors, so the robot won't crash into the wall, and the maze in this case represent the complicated environment in the world.

Phenotype:

We dedicate Individual class as Phenotype, which contains one genotype.

Genotype:

In individual class (Phenotype), we have one chromosome and its type is int[] with binary value 0 and 1. We call 0 and 1 value as gene.

Fitness:

The fitness score is a number that represents how good a solution to the problem this individual is. The fitness of each solution is calculated by the route. We initially set the route we want and calculate the route provided by chromosomes. In each route, we calculate whether this route passes through the route we set, and it will be scored.

Crossover:

We choose the highest fitness solution by ***tournament selection***, which is the higher an individual's fitness the greater chance that the individual will be chosen for ***single point crossover*** and mutate for the next generation. In this crossover method, a single point in the

chromosome is picked at random, then any genetic information before that point comes from parent A, and any genetic information after that point comes from parent B. The final solution will be found until the number of generations reach the maximum we set.

Initialization:

The maze object we've created uses integers to represent different terrain types: 1 defines a wall; 2 is the starting position, 3 traces the best route through the maze, 4 is the goal position and 0 is an empty position that the robot can travel over but isn't on the route to the goal.

This code contains a public method to get the start position, check a position's value and score a route through the maze.

The scoreRoute method is the most significant method in the Maze class; it evaluates a route taken by the robot and returns a fitness score based on the number of correct tiles it stepped on. The score returned by this scoreRoute method is what we'll use as the individual's fitness score in the GeneticAlgorithm class' calcFitness method.

Expression:

We define one chromosome as one solution. Each solution indicates a series of possible actions by binary code, such as

10011101110001001110100101101010000001001111001101010110010101111100011010110100100001010101110100010110100111001011000110010.

Our robot will have four actions: do nothing, move forward one step, turn left and turn right. These can be represented in binary as:

- "00": do nothing
- "01": move forward
- "10": turn left
- "11": turn right

Execution

We use *IntelliJ* as our IDE for this project, and the framework we use is *Maven*, we have five different packages: *controller, helper, impl, repository and robot_main*. We put the robot controller under *controller* package, put maze generator under *helper* package, put our Interfaces under *repository* package, put our implements for Interfaces under *impl* package and put our Main class under *robot_main* package, we also have a maze folder to store our maze and a test package to store our tests. **To execute our project**, you need to run the Main class under *robot_main* package, then it will call the controller class to run our program, you also need to input the Maze name in the console, for example, input "*Maze1"*, we have 4 different Mazes, if you want to run the default Maze, you can just press *Enter*.

Classes

GeneticAlgorithmImpl Class:

Inherited from GeneticAlgorithm Interface. Implement Genetic Algorithm.

IndividualImpl Class:

Inherited from Individual Interface. About Individuals, including chromosome and fitness. It represents to one route solution in this problem.

MazeImpl Class:

Inherited from Maze Interface. Define the Maze which contains the route and score the fitness.

PopulationImpl Class:

Inherited from Population Interface. It's related to the array of Individual.

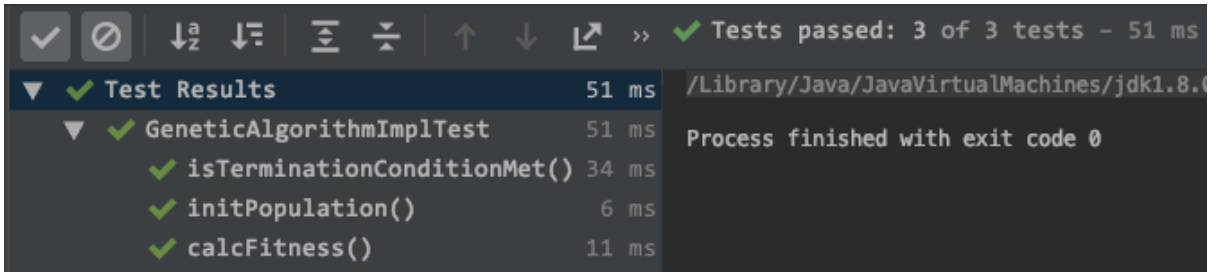
RobotImpl Class:

Inherited from Robot Interface. About all sensors and make the decisions of actions.

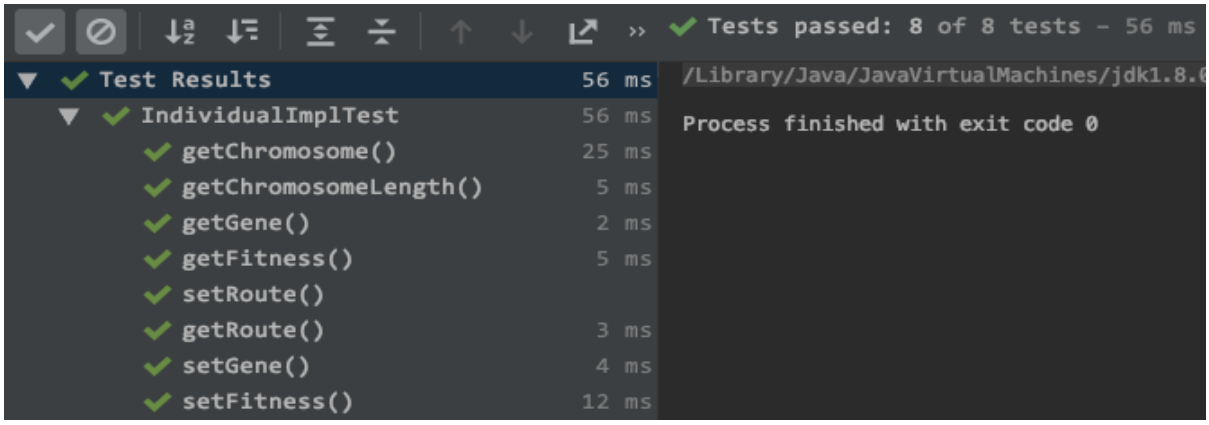
Experiment

Test:

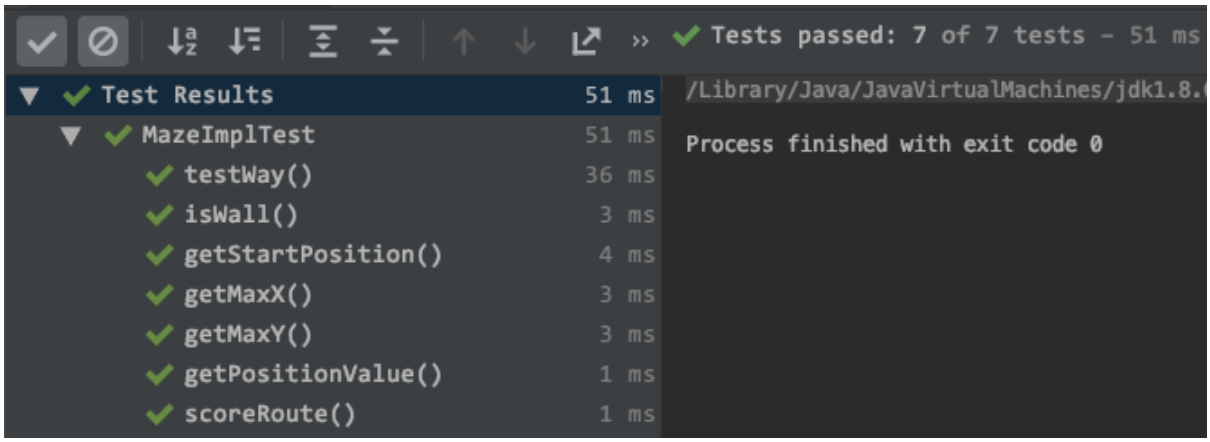
Genetic Algorithm



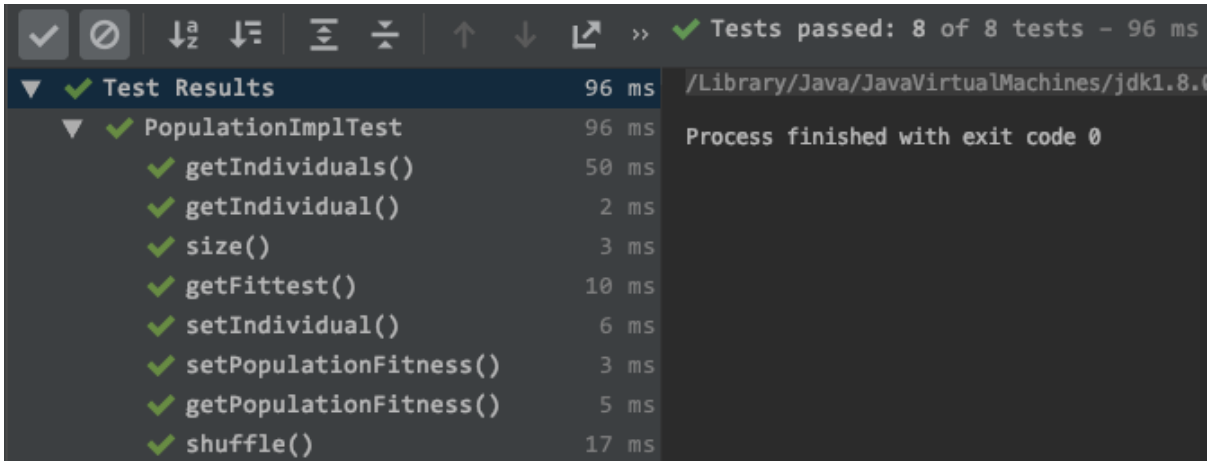
Individual



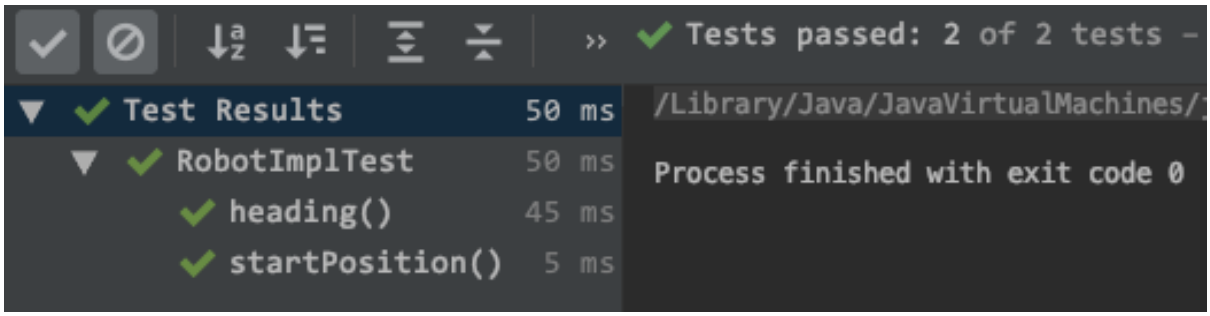
Maze



Population



Robot



Cases:

Maze1:

0	0	0	0	1	0	1	3	2
1	0	1	1	1	0	1	3	1
1	0	0	1	3	3	3	3	1
3	3	3	1	3	1	1	0	1
3	1	3	3	3	1	1	0	0
3	3	1	1	1	1	0	1	1
1	3	0	1	3	3	3	3	3
0	3	1	1	3	1	0	1	3
1	3	3	3	3	1	1	1	4

(9x9)

Fittest Route

Different Generation Times

maxGenerations=10 maxGenerations=100 maxGenerations=1000

According to the result above, different generation times correspond to different fittest route result. When we generate it 10 times, the fittest route stop in the middle. When we increase the generation times to 100, it has better solution, but still not reach our goal position. When we increase the generation times to 1000, it reached our goal position.

Different Population Size

Population Size=10 Population Size=100

According to the result above, different population sizes also correspond to different fittest route result. When the population size is 10, the fittest solution is not reach our goal position. When we increate the population size to 100, we reach our goal position.

On the basis of our experiment above, we decided to setup our max generation times as 1000, and the number of population size as 200.

Chromosome of the best solution (Fittest=30.0):

1001101100010111111011000100101001100110111101110100010001110111000001001011011001010111011001001100010100010111010111000100010

Maze2:

0	1	3	3	2
0	1	3	1	1
1	1	3	1	0
3	3	3	1	0
4	1	1	1	0

(5x5)

Best solution (8.0):

1111101001100100111011000100111010101010011101111101010101010110000010101111001011011000101001101000011110110111101001001111011010

Maze3:

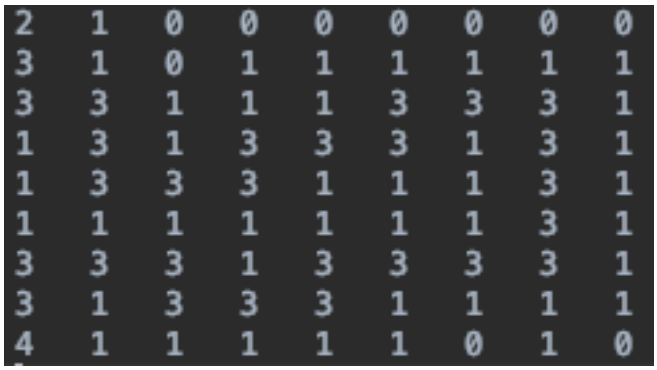
2	1	1	4
3	1	3	3
3	1	3	1
3	3	3	1

(4x4)

Best solution (9.0):

00110101010111010001111001010001000100011010001111110100011001100101100000100111001110000110111011111001000001101100011011100111

Maze4:



(9x9)

Best solution (28.0):

10011111011101101110111000100110000010010011011110000110010001110100100010110100100111100110011000010101111001010101011111010100

In above experiment, we can see the fittest route result match the route we setup in the Maze, and the number of fitness is equal to the number of 3 and 4 in the Maze.

Conclusion

Genetic algorithms can be used to design sophisticated controllers, which may be difficult or time consuming for a human to do manually. By giving the robot a maze and a preferred route, a genetic algorithm can be applied to find a controller that can use the robot's sensors and successfully navigate the maze.

Reference

Lee Jacobson, Burak Kanber. *Genetic Algorithms in Java Basics.*