INFO 6205 Final Project – Generic Algorithm for *Robot Controller*

Group 215

Group Members:

PinHo Wang

PeiChun Tseng

Xinxin Huang

Instructor: Robin Hillyard

Implements

Problems:

The problem is to design the robot controller to walk through the maze and arrive the destination in the end without crashing into the wall. The robot can be controlled by six different direction sensors, three on the front, one on the left, one on the right and one on the back. Each route can be one solution and can be scored, our purpose is to find the fittest solution by comparing the scores until it reaches the termination we set.

We choose the highest fitness solution in each generation and change it by crossover and mutate for the next generation. The final solution will be found until the number of generations reach the maximum we set.

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 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.

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.

The calcRoute method in the 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 calcRoute method is what we'll use as the individual's fitness score in the GeneticAlgorithm class' calcFitness method. When robot enter position 3 or position 4, it can score one point (fitness).

Expression:

We define one chromosome as one solution. Each solution indicates a series of possible actions by binary code, such as 10011110…11. The definition of actions shows below.

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

Phenotype:

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

Code

GeneticAlgorithm class: The main GA implements.

Individual class: Details about individuals, including chromosome and fitness. It means one route solution in this problem.

Population class: Details about populations, including individuals array.

Maze class: Define the maze that contains the route and score the fitness.

Robot class: Details about all sensors and make the decision of the actions.

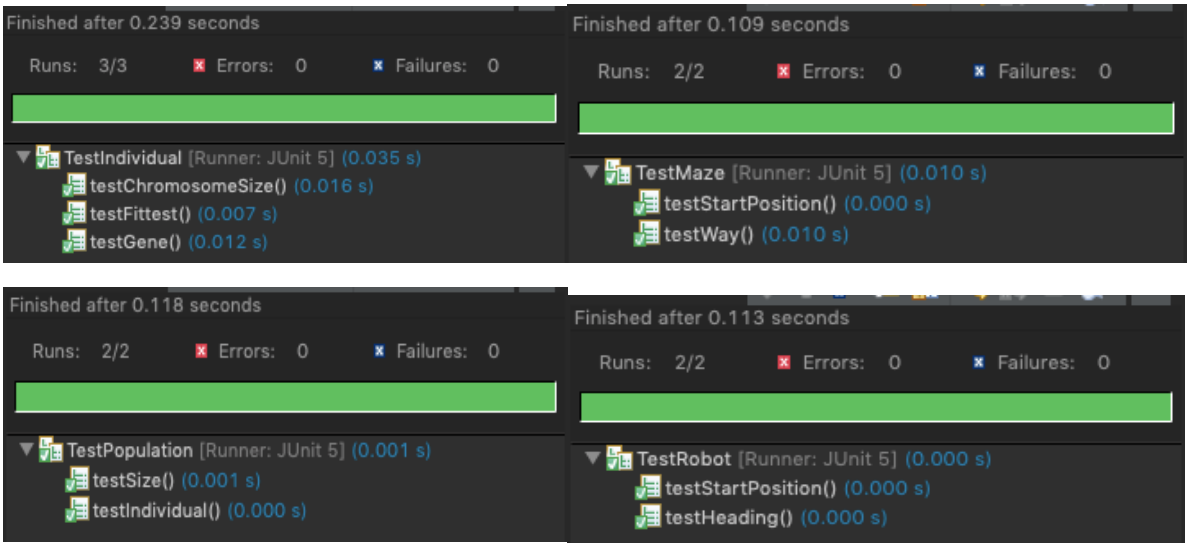
Execution

In this project, we use *eclipse* as our IDE and we put codes into different package separately, including helper, main, controller, objects and test. The main function is in the Main.java in main package, Controller.java is in controller package, all the test cases are in the test package and all the class we create, including abstract class, are in objects package.

To execute, just run the Main.java and it will use the Controller.java (in controller package) to run our whole GA program. It will print the fittest value and fittest chromosome for each generation. In the end, it will print the route of final fittest solution.

Experiment

Test:



Cases:

Maze1 Maze(9X9)

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

Fittest Route

maxGeneration = 10

```
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 1 1 0
0 0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

maxGeneration = 50

```
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 1 1 0
1 1 1 0 1 0 0 0 0
1 0 1 1 1 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

maxGeneration = 500

```
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 1 1 0
1 1 1 0 1 0 0 0 0
1 0 1 1 1 0 0 0 0
1 1 0 0 0 0 0 0 0
0 1 0 0 1 1 1 1 1
0 1 0 0 1 0 0 0 1
0 1 1 1 1 0 0 0 1
```

As the results above, different times of generation correspond to different fittest result. When it comes to 10 generations, the fittest route stops in the middle of the maze, then 50 case has a better solution but doesn't still reach the goal position. Finally, when it comes to 500 times, the goal position reaches.

Then, we set the maxGeneration to 500 and test the effect of number of the populations.

# of populations = 10

```
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 1 1 0
0 0 1 0 1 0 0 0 0
0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

# of populations = 50

```
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1 0
0 0 0 0 1 1 1 1 0
1 1 1 0 1 0 0 0 0
1 0 1 1 1 0 0 0 0
1 1 0 0 0 0 0 0 0
0 1 0 0 1 1 1 1 1
0 1 0 0 1 0 0 0 1
0 1 1 1 1 0 0 0 1
```

The result shows that when the number of populations is 10, the fittest solution doesn't reach to the goal position, but the case of 50 does.

In the end, we set maxGeneration as 1000 and number of populations as 2000

Chromosome of the best solution (Fittest = 30.0):
1111110111000100101010111011110101011011011111011010001100100011010010100011001001101111001100
1000011010111000101001000000101100 1

Maze2

Maze(5X5)

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

Fittest Route

```
0 0 1 1 1
0 0 1 0 0
0 0 1 0 0
1 1 1 0 0
1 0 0 0 0
```

Best solution (8.0):
1111100001010110011001100110110011100111010011111111011000011110100100000101010110110010110101111101101011000111101011100100100 1
The result is quite corresponding the route we expect.

Maze3

Maze(4X4)

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

Fittest Route

```
0 0 0 1
1 0 1 1
1 0 1 0
1 1 1 0
```

Best solution (8.0):
1001111100101011101110011101101100000101001011111010011101000100011101000100011001100110101110100101011001110110000000010010010 0

Maze4

Maze(4X4)

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

Fittest Route

```
0 0 1 1 1 0
0 0 0 0 1 0
0 1 1 1 1 0
0 1 0 0 0 0
0 1 0 1 1 1
1 1 1 1 0 1
```

Best solution (13.0): chromosome
0010101111100101001001010001011001110101111111111111011101100110010011011010010010001010011101111100010101110111101001000111011 1

In this experiment, we can see the best rout's result match the Maze4's route and the best solution's fitness is 13 that is equal to the number of 3 and 4 in Maze4.