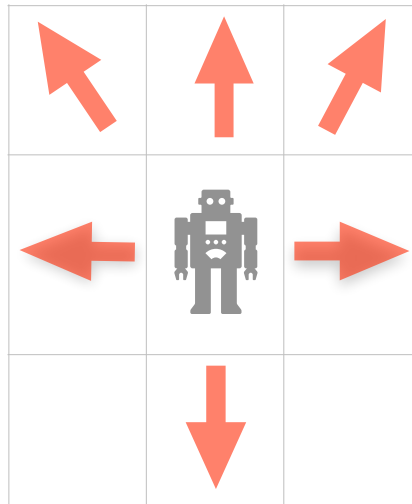# INFO6205 Final Project Report

Problem Description

The problem we are going to solve is   designing a robot using sensor to through a maze successfully. The robot who has four actions: move one-step forward, turn left, turn right and stay still. The robot also has six sensors: front Sensor, front-left Sensor, front-right Sensor, left Sensor, right Sensor and back Sensor.


The maze the robot will explore contains walls and the route we preset, which we want the robot to follow. In our project, our purpose is to automatically program a robot controller with six sensor so that it doesn't crash into walls. The robot use its sensor to detect whether there is a wall adjacent to the sensor. For example, the robot will detect a wall in front of it using the front Sensor.

Genetic Algorithm Design

Environment Design:

We created maze object use integer to represent different situations: 1 defines a wall; 2 is the starting position, the series of 3 represents the best route through the maze, 4 is the ending position and 0 is am empty position which robot may travel but it is not belonged to the best route to the ending position.

1	1	1	1	1	1	1
1	0	3	3	3	2	1
1	1	3	1	1	1	1
1	1	3	0	1	0	1
1	0	3	1	1	1	1
1	1	3	3	3	4	1
1	1	1	1	1	1	1
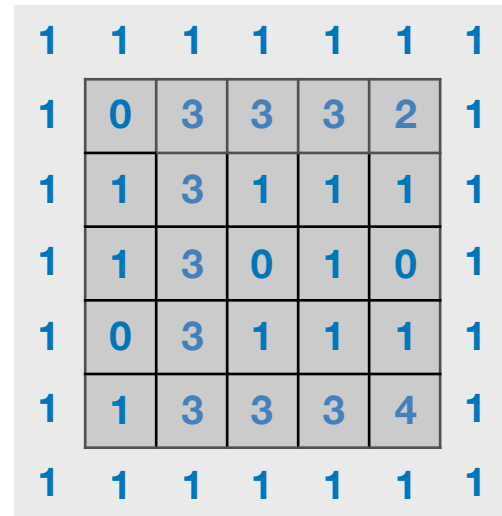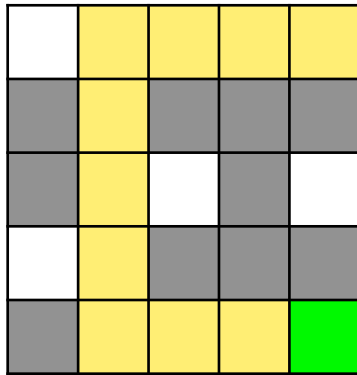
Robot Design:

The robot has four actions: stand still, move forward, turn left and turn right. And they can be represented in binary as: "00" represents standing still, "01" means moving forward, "10" is turning left and "11" means turning right.

The robot use sensor value and robot heading direction to make the direction of next move.

Situation 1: when the sensor value is 1, meaning the robot move forward.
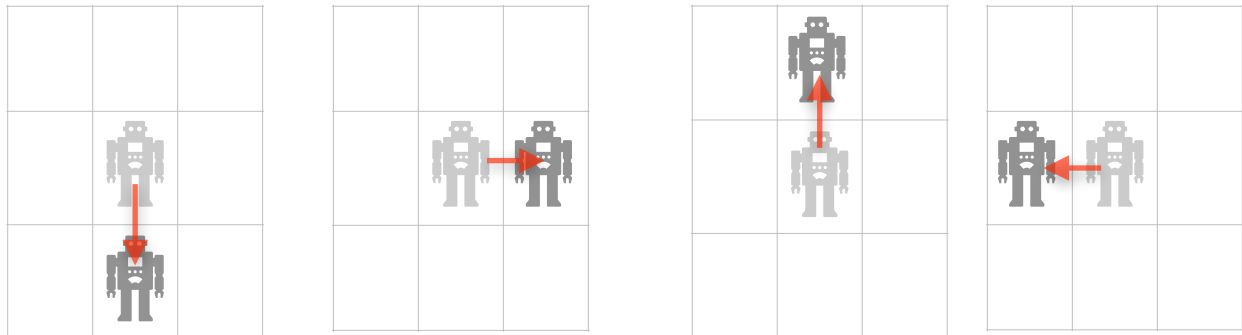
Situation 2: when the sensor value is 2, meaning the robot make clockwise turning
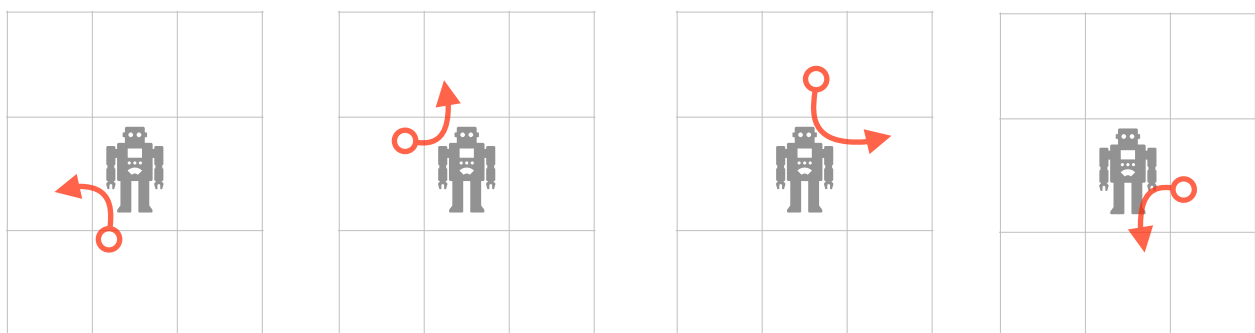
Situation 3: when the sensor value is 3, meaning the robot make anti-clockwise turning
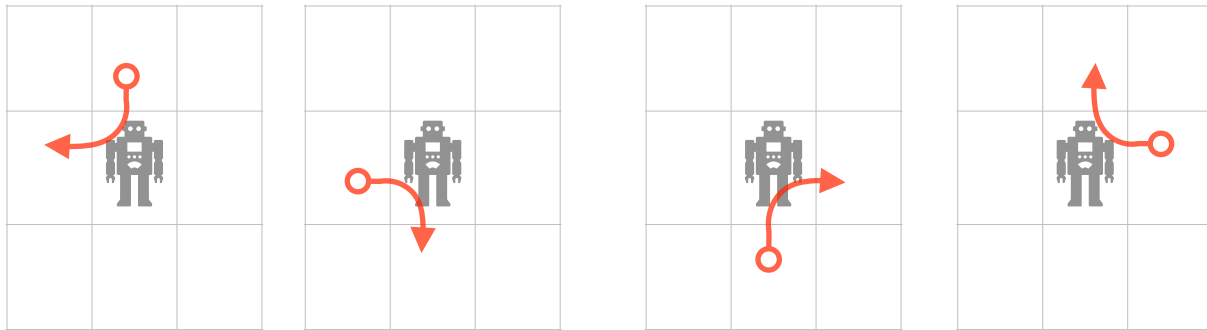
Gene Design:

1. Gene order:

The robot has six sensors, which means it will give us 64 possible combinations of sensor inputs. We use 2 bits to encode each action. Since genetic algorithm chromosomes are easiest to manipulate as arrays, our chromosome will be a bit array of length 128. Therefore, we can use chromosome whose length is 128 to represent each action of robot.

As the gene inside a robot sensor should be able to store the 64 combinations of sensor inputs. We encode each action with 2 bits. So 128 gene positions should be included for each individual.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	....	127	128
1	1	0	0	1	0	0	1	1	0	1	0	0	1		1	1

We need a binary representation of the robot controller's complete instruction set for all possible combinations of inputs.

According to the discussion above, the robot will have 4 different actions: stand still, move forward, turn left and turn right. These can be represented in binary as:

| 0 | 0 | → **Move Forward**

| 1 | 1 | → **Turn Right**

| 0 | 1 | → **Stand Still**

| 1 | 0 | → **Turn Left**

2. Sensor decide the position in the chromosome

The "sensors" column represents the sensors' bitfield, which maps to a chromosome position after you convert the bitfield to decimal. Once you've converted the sensors' bitfield to decimal, you can place the desired action at the corresponding location in the chromosome.

Sensors						Action	Chromosome
B	R	L	FR	FL	F		
0	0	0	0	0	0	"11"	11001001101001....11
0	0	0	0	0	1	"00"	11001001101001....11
0	0	0	0	1	0	"10"	11001001101001....11
0	0	0	0	1	1	"01"	11001001101001....11
0	0	0	0	0	0	"10"	11001001101001....11
0	0	0	1	0	1	"10"	11001001101001....11
0	0	0	1	1	0	"01"	11001001101001....11
1	1	1	1	1	1	"11"	11001001101001....11

3. Fitness Function Design:

According to the discussion above, we have preset a route for the robot, when the robot follow the route preset.
Each correct move ⟶ + 1 score

Evolution Process

Initializations:

1. Initialization of individual:

We set up four different population number of each generation, which is 50, 100, 200, 300. The 128 genes for each individual are inputted randomly with 0 and 1.

1	2	3	4	5	6	7	8	9	10	11	12	13	14		127	128
1	1	0	0	1	0	0	1	1	0	1	0	0	1	....	1	1

2. Initialization of the maze:

We have preset three different mazes. They are 9*9 matrix, 5*5 matrix, 9*5 matrix.
We set the start position of robot is the top left corner of these matrixes.

Maze 1

Maze 2

Maze 3
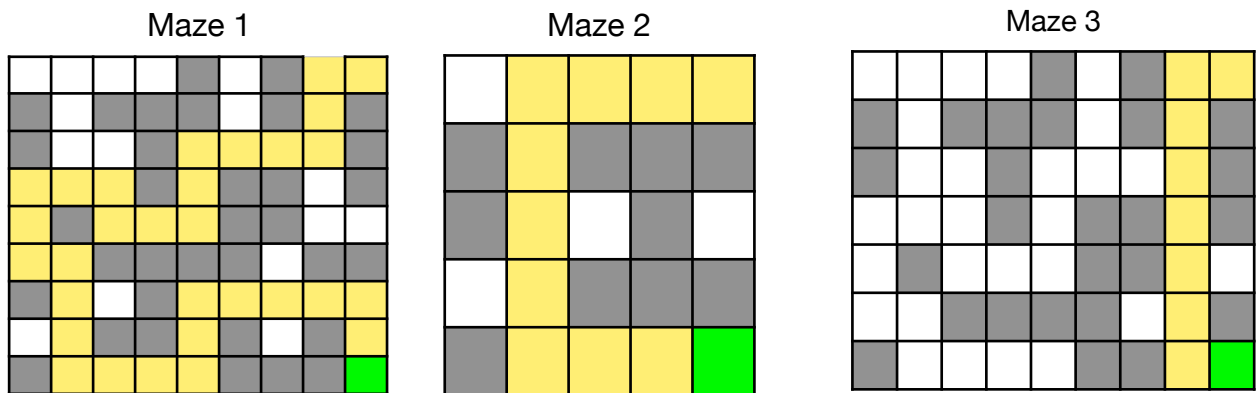
3. Apply Genetic Algorithm

First Generation Initialization

Evaluate the whole population using the maze, return the total fitness of the whole population.keep track of the

Start evolution loop

Find the fittest individual from the population, if the route has been found, print the route.

Apply the crossover and mutation and evaluate the current

After multiple generations, get the best solution.

Results & Conclusions

It can be predicted that the fitness of the first generation will be bad, because the chromosome is generated randomly. Therefore, the robot will get lower score, and it can not find the route of the maze. However, some individual can get the higher score although their genes are generated randomly. The higher score they get, the higher possibility they will be used to propagate next generation. After multiple generation generated, we can find the fittest individual, whose fitness is the highest, and get the highest score.

According to the problem description, we preset a maze with a preferred route. By using a genetic algorithm, a robot can navigate the maze successfully with the help of sensor. This can be done by assigning each sensor an action in the individual's chromosome encoding. By making small random changes with crossover and mutation, guided by the selection process, better performance can be found.

Evidence of Analysis

For Maze 2, we set the parameters of GA as following:

maxGenerations = 1000
populationSize = [10, 200]
mutationRate = 0.05
crossoverRate = 0.95
elitismSize = 2
tournamentSize = 10
simulationTimes = 50

Analysis the Influence of Population Size
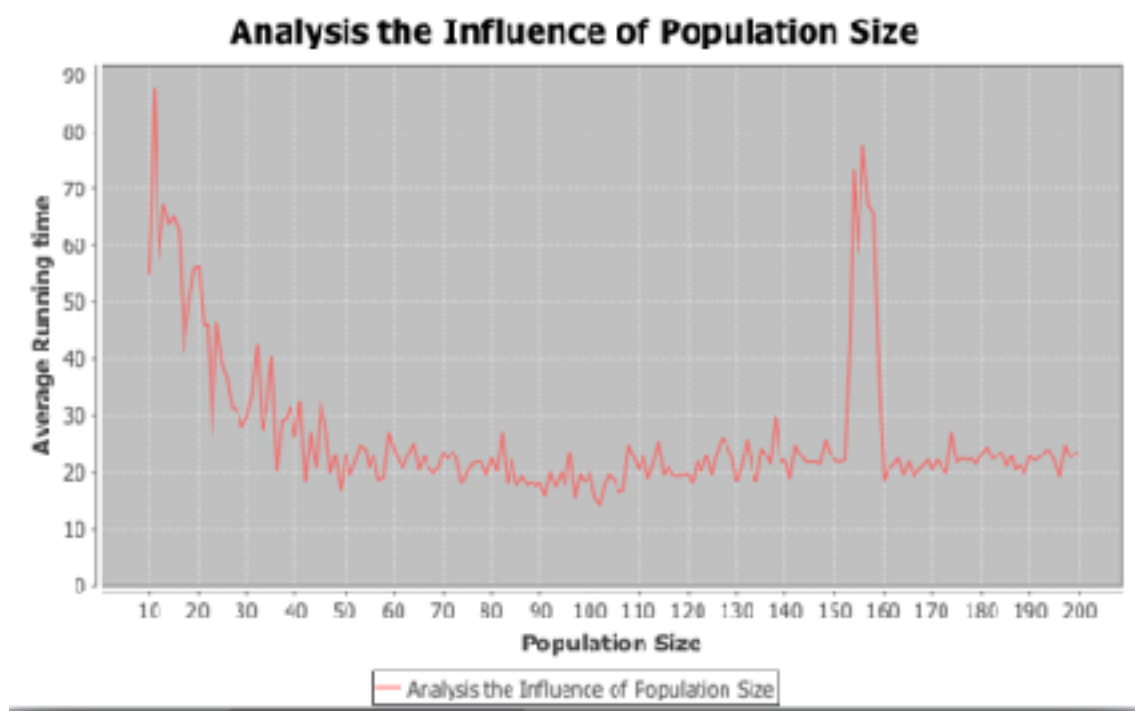
For Maze 2, we set the parameters of GA as following:

maxGenerations = 1000
populationSize = [10, 500]
mutationRate = 0.05
crossoverRate = 0.95
elitismSize = 2
tournamentSize = 10
simulationTimes = 50

Analysis the Influence of Population Size

— Analysis the Influence of Population Size

From the above picture, we can concluded that for maze 2, when the population is 150, the average running time for generating best individual have better performances.

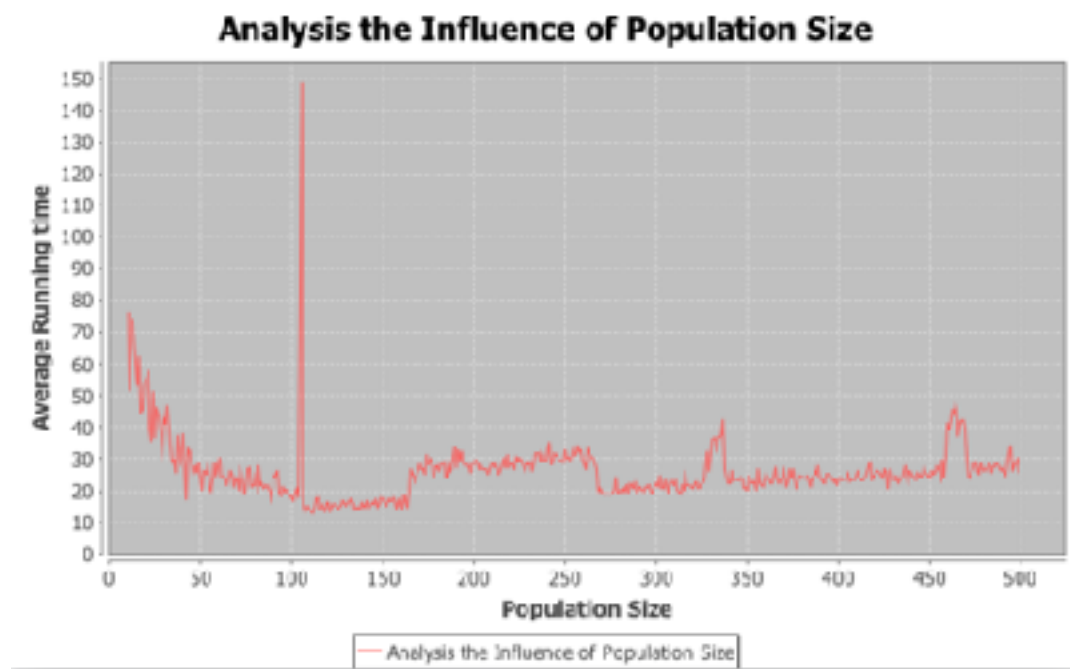Then we set the population for maze 2 is 150, and we did the analysis about the crossoverRate.

For Maze 2, we set the parameters of GA as following:

maxGenerations = 1000
populationSize = 150
mutationRate = 0.05
crossoverRate = [0,1]
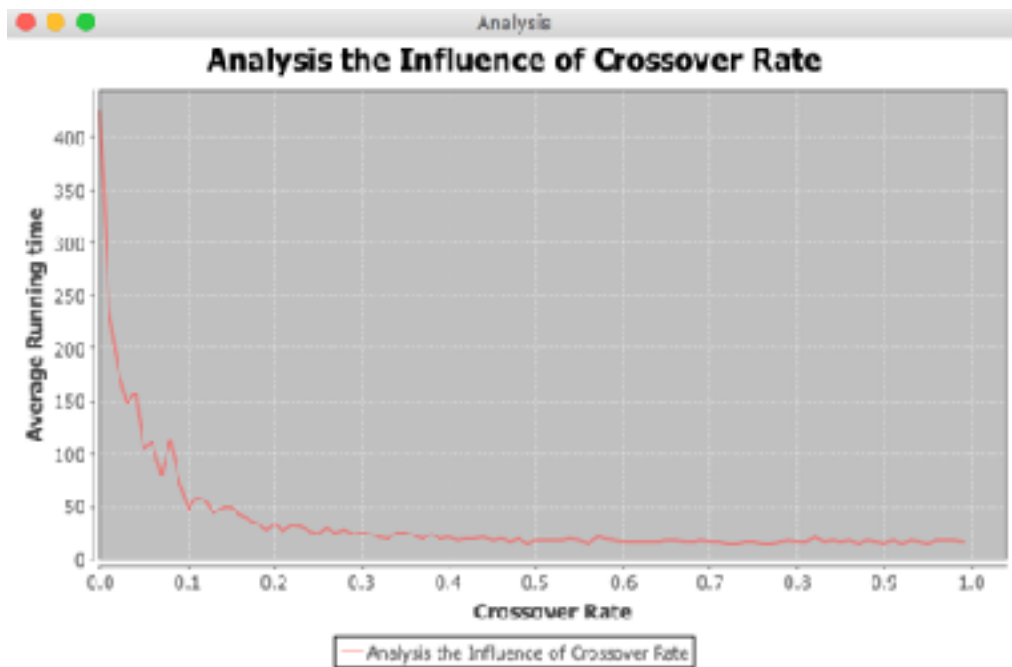elitismSize = 2
tournamentSize = 10
simulationTimes = 50

Analysis the Influence of Crossover Rate

From the above picture, we can concluded that when the crossoverRate is 0.95 for maze 2, the average running time have better performances.

Therefore, for maze 2, the robot controller has better performance when the population is 150 and the crossoverRate is 0.95.

Population Size	Mutation Rate	Crossover Rate	Elitism Size	Tournament Size	Maximum Generaions	Simulation Times	Route Found Count	Average Generations	Average Running Time(ms)
300	0.05	0.9	2	10	200	100	100	8	23.62
300	0.05	0.9	2	10	100	100	100	10	25.15
300	0.05	0.9	2	10	50	100	100	8	20.94
300	0.05	0.9	2	10	30	100	99	9	22.67
200	0.05	0.9	2	10	200	100	100	10	19.34
200	0.05	0.9	2	10	100	100	100	11	19.73
200	0.05	0.9	2	10	50	100	100	12	19.94
200	0.05	0.9	2	10	30	100	96	9	16.06

Population Size	Mutation Rate	Crossover Rate	Elitism Size	Tournament Size	Maximum Generaions	Simulation Times	Route Found Count	Average Generations	Average Running Time(ms)
100	0.05	0.9	2	10	200	100	100	18	17.14
100	0.05	0.9	2	10	100	100	98	18	17.85
100	0.05	0.9	2	10	50	100	98	15	14.94
100	0.05	0.9	2	10	30	100	92	13	12.35
50	0.05	0.9	2	10	200	100	100	30	16.24
50	0.05	0.9	2	10	100	100	99	28	15.68
50	0.05	0.9	2	10	50	100	84	19	11.19
50	0.05	0.9	2	10	30	100	62	11	8.32
300	0.05	1.0	2	10	200	100	100	9	24.12
300	0.05	0.95	2	10	200	100	100	8	20.47
300	0.05	0.85	2	10	200	100	100	9	22.83
300	0.05	0.80	2	10	200	100	100	9	22.11
200	0.05	1.0	2	10	200	100	100	10	17.92
200	0.05	0.95	2	10	200	100	100	10	17.11
200	0.05	0.85	2	10	200	100	100	11	20.04
200	0.05	0.80	2	10	200	100	100	11	18.15
100	0.05	1.0	2	10	200	100	100	16	15.6
100	0.05	0.95	2	10	200	100	100	19	17.86
100	0.05	0.85	2	10	200	100	100	17	16.01
100	0.05	0.80	2	10	200	100	100	17	18.64
50	0.05	1.0	2	10	200	100	100	30	16.66
50	0.05	0.95	2	10	200	100	100	31	16.77
50	0.05	0.85	2	10	200	100	100	35	18.66
50	0.05	0.80	2	10	200	100	100	39	24.12
300	0.05	0.95	3	10	200	100	100	8	22.14
300	0.05	0.95	4	10	200	100	100	8	22.14

Population Size	Mutation Rate	Crossover Rate	Elitism Size	Tournament Size	Maximum Generaions	Simulation Times	Route Found Count	Average Generations	Average Running Time(ms)
300	0.05	0.95	5	10	200	100	100	9	23.19
200	0.05	0.95	3	10	200	100	100	11	19.89
200	0.05	0.95	4	10	200	100	100	14	24.62
200	0.05	0.95	5	10	200	100	100	13	25.92
100	0.05	0.95	3	10	200	100	100	17	17.1
100	0.05	0.95	4	10	200	100	100	21	19.04
100	0.05	0.95	5	10	200	100	100	23	21.29
50	0.05	0.95	3	10	200	100	96	35	15.94
50	0.05	0.95	4	10	200	100	96	28	17.82
50	0.05	0.95	5	10	200	100	98	38	20.03
300	0.05	0.95	2	5	200	100	100	9	23.38
300	0.05	0.95	2	50	200	100	100	7	22.99
300	0.05	0.95	2	100	200	100	100	7	27.59
300	0.05	0.95	2	200	200	100	100	8	49.07
200	0.05	0.95	2	5	200	100	100	14	25.12
200	0.05	0.95	2	50	200	100	100	9	22.46
200	0.05	0.95	2	100	200	100	100	10	27.11
100	0.05	0.95	2	5	200	100	100	21	19.99
100	0.05	0.95	2	20	200	100	100	16	16.49
100	0.05	0.95	2	50	200	100	100	15	18.29
50	0.05	0.95	2	5	200	100	98	34	21.37
50	0.05	0.95	2	20	200	100	100	27	15.65
50	0.05	0.95	2	30	200	100	100	27	16.6

Evidence of Running

Program Outputs:

1. Map and Route

2. Evolution Process

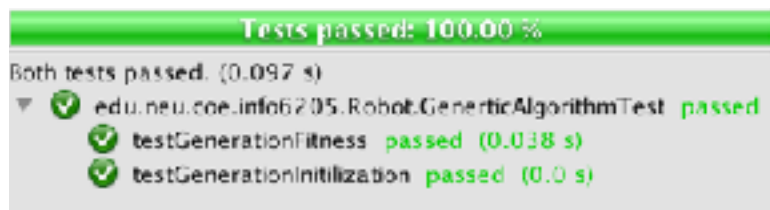### 2. Generations for Finding the Route

Unit Tests

Maze Tests:

Tests passed: 100.00 %

All 4 tests passed. (0.067 s)
- ▼ ✅ edu.neu.coe.info6205.mazeTest.MazeTest passed
  - ✅ maze_isWall passed (0.001 s)
  - ✅ maze_getMaxXAndgetMaxY passed (0.0 s)
  - ✅ maze_scoreRoute passed (0.001 s)
  - ✅ maze_startPoint passed (0.0 s)

Genetic Algorithm Tests:

Tests passed: 100.00 %

Both tests passed. (0.097 s)
- ▼ ✅ edu.neu.coe.info6205.Robot.GenerticAlgorithmTest passed
  - ✅ testGenerationFitness passed (0.038 s)
  - ✅ testGenerationInitilization passed (0.0 s)

Robot Tests:

Tests passed: 100.00 %

All 4 tests passed. (0.077 s)
- ▼ ✅ edu.neu.coe.info6205.Robot.RobotTest passed
  - ✅ testStartPosition passed (0.02 s)
  - ✅ testClockwise passed (0.0 s)
  - ✅ testAntiClockwise passed (0.0 s)
  - ✅ testmakeNextAction passed (0.0 s)