Zagazig University
Faculty of Engineering
Computer and Systems Engineering
Dept.
AI Mini Project Report
year : 2017

Robot Emulator

By,

Ahmed Abd Elbadee Elsayed
Mostafa Elsayed Gohary
Amira Ibrahim Soliman
Nourhan Mansour Mohamed
Yasmeen Ahmed Abd Elbaset

Supervisor,

Dr. Mohamed Amal
Eng.Amr Ahmed Zamel

This document is the report concerning the Mini-Project done in the winter semester 2016 by our group. The group was composed of five students who studied Computer and Systems Engineering on the fourth academic year at Zagazig University, Faculty of Engineering.

Zagazig, Friday, February 3, 2017.

Participant(s):

Name	ID	Task(s)	Signature
1 أحمد عبد البديع السيد		Fitness Function, Algorithm Class	
2 مصطفى السيد جوهرى		Mutation, GUI	
3 أميرة ابراهيم سليمان		Data Implementation	
4 نورهان منصور محمد		Initials Generation	
5 ياسمين أحمد عبد الباسط		Crossover	

Abstract:

We have an environment with some conditions such as wind, rain, rubble...etc, and we need a robot with the best characteristics so it can handle these conditions and do specific functions.

For this matter we used the *Genetic Algorithm* so we can get the best robot characteristics against the conditions of our environment.

1. Introduction

Making a robot that can function in an effective way is a problem for a lot of individuals and companies who do not have enough data about the best robot characteristics against a certain environment conditions.

2. Proposed Algorithms

To solve that kind of problem we need to use an algorithm that can search a lot of solutions and calculate their effectiveness against the environment conditions so we can get the best possible solution, so that we used the *Genetic Algorithm.*

3. Algorithm Implementation

We have implemented the algorithm in *Python* programming language and to implement the genetic algorithm for this problem we had to follow some steps:

3.1 Data Implementation

The chromosome in our application is expressed as a string of binary bits, each two bits refer to a characteristic for the robot.

So first we needed some data for the algorithm to work with, we have two kinds of data:

3.1.1 Robot characteristics data

For the algorithm to create initial solution, it needs some data to work with, we have implemented each characteristic as a list of lists, the bigger list contains different forms of this characteristic and each list inside the bigger list contain the actual form, the binary code and the implementation cost of this form *(Appendix A.1).*

3.1.2 Environment conditions data

For the algorithm to calculate the effectiveness of the chromosome against the environment, it needs the data of the environment conditions.

We have implemented these data as classes, each class expresses an environment and has functions that return the best characteristics that can fit in this environment *(Appendix A.2).*

3.2 Initials Generation

For the creation of the first generation, we have created a class, each function in this class creates a characteristic randomly from the available data and adds it the chromosome string *(Appendix A.3).*

3.3 Fitness Function

We need to calculate the effectiveness or the fitness of each chromosome against the environment and the fitness function is the responsible for that.

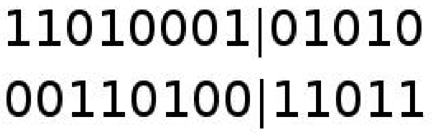
The fitness function is a class, each function in this class is responsible for evaluating a single characteristic of the chromosome against the environment and then returns the fitness of it *(Appendix A.4)*

3.4 Selection

After calculating the fitness, we need to select the good chromosomes and discard the bad ones, so we select the most two fittest chromosomes and discard the rest.

3.5 Crossover

Crossover class is responsible for creating another two chromosomes from the two fittest chromosome which were chosen from the previous generation, by replacing parts of the two chromosomes with each other and creating new two chromosomes *(Appendix A.5)*



11010001|01010
00110100|11011

Figure 3.1

3.6 Mutation

The Mutation class is responsible for mutating a single bit in each of the four chromosomes so it can generate four new chromosomes with the previous four so we have eight chromosome in the new generation to be work with *(Appendix A.6)*

3.7 Code Running

There are two ways to end the algorithm, the first is by determining the number of generations to be produced and then return the fittest chromosome in the last generation, and the second is by checking the maximum fitness in each generation and returning a certain chromosome with a certain fitness that specified earlier *(Appendix A.10)*

3.8 GUI

For the ease of running the application, the gui contains the different environments that the code can work with and some fields for displaying the characteristics of the fittest chromosome.

It also contains an illustration for the maximum fitness that each generation had.

The user just picks an environment and presses *start* for the algorithm to run *(Appendix A.8)*

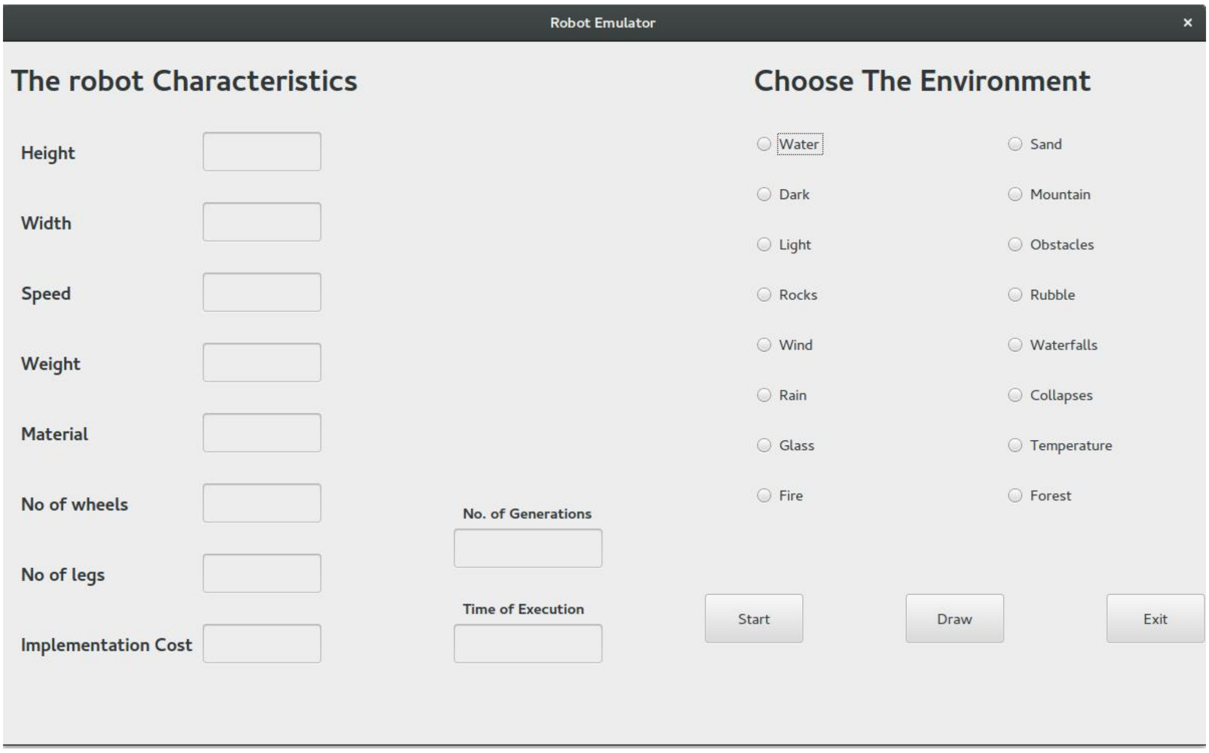
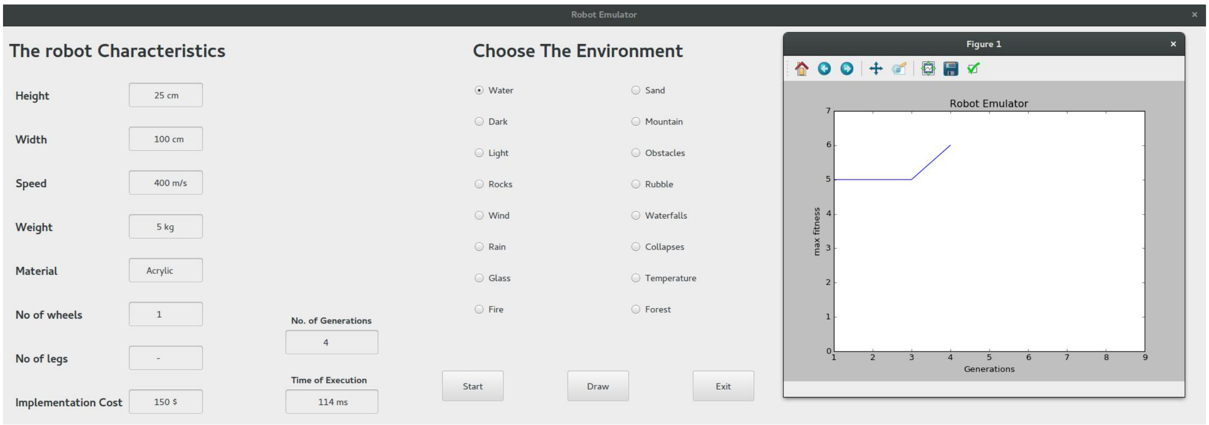


Figure 3.2

4. Results

Here are some figures and statistics for different cases when running the algorithm:



Robot Emulator

The robot Characteristics

Height	25 cm
Width	100 cm
Speed	400 m/s
Weight	5 kg
Material	Acrylic
No of wheels	1
No of legs	-
Implementation Cost	150 $

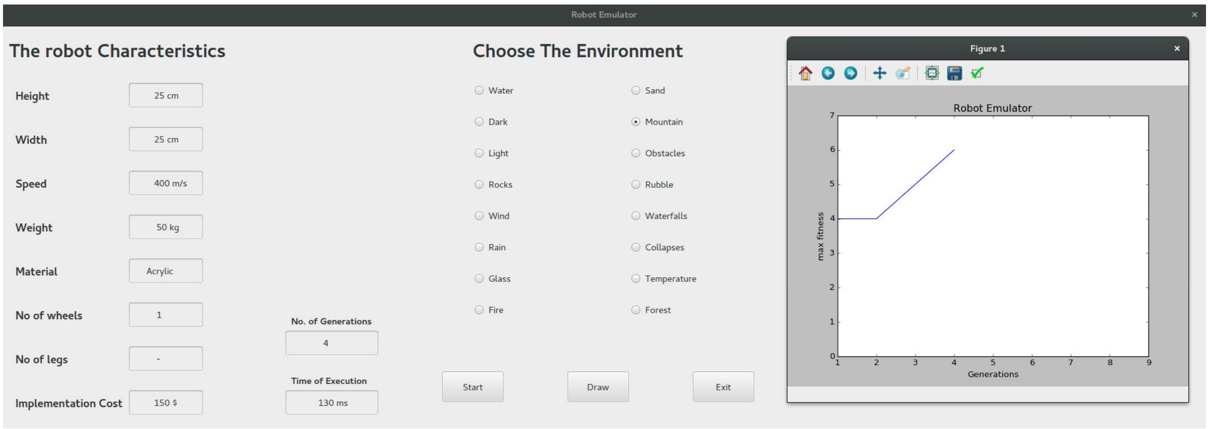
No. of Generations: 4

Time of Execution: 114 ms

Choose The Environment

● Water ○ Sand
○ Dark ○ Mountain
○ Light ○ Obstacles
○ Rocks ○ Rubble
○ Wind ○ Waterfalls
○ Rain ○ Collapses
○ Glass ○ Temperature
○ Fire ○ Forest

Figure 4.1



Robot Emulator

The robot Characteristics

Height	25 cm
Width	25 cm
Speed	400 m/s
Weight	50 kg
Material	Acrylic
No of wheels	1
No of legs	-
Implementation Cost	150 $

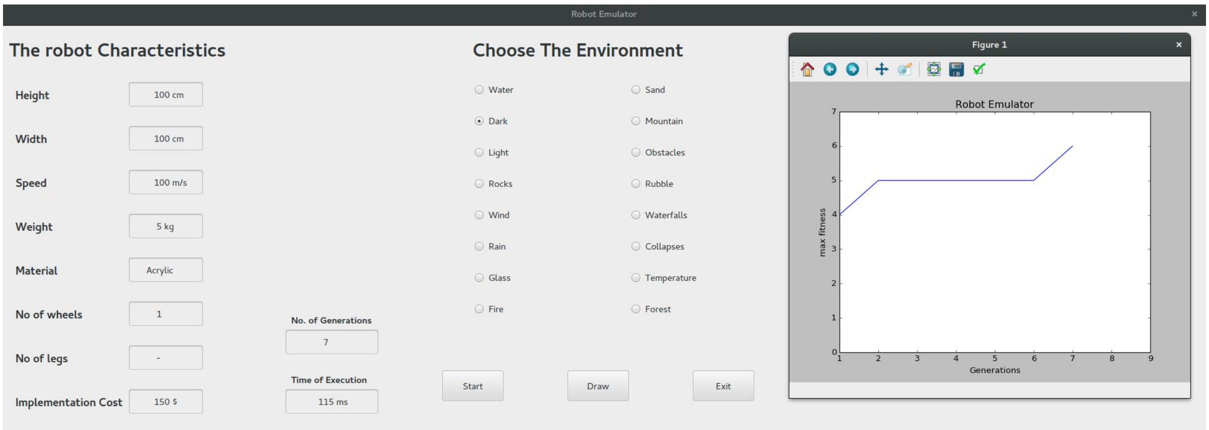
No. of Generations: 4

Time of Execution: 130 ms

Choose The Environment

○ Water ○ Sand
○ Dark ● Mountain
○ Light ○ Obstacles
○ Rocks ○ Rubble
○ Wind ○ Waterfalls
○ Rain ○ Collapses
○ Glass ○ Temperature
○ Fire ○ Forest

Figure 4.2



Robot Emulator

The robot Characteristics

Height	100 cm
Width	100 cm
Speed	100 m/s
Weight	5 kg
Material	Acrylic
No of wheels	1
No of legs	-
Implementation Cost	150 $

No. of Generations: 7

Time of Execution: 115 ms

Choose The Environment

○ Water ○ Sand
● Dark ○ Mountain
○ Light ○ Obstacles
○ Rocks ○ Rubble
○ Wind ○ Waterfalls
○ Rain ○ Collapses
○ Glass ○ Temperature
○ Fire ○ Forest

Figure 4.3

Appendix A (code)

A.1 Data

```python
# Characteristic = [["actual", "code", implementation cost], ...]

height = [["25", "00", 10], ["500", "01", 20], ["75", "10", 30], ["100", "11", 40]]

width = [["25", "00", 10], ["50", "01", 20], ["75", "10", 30], ["100", "11", 40]]

no_of_wheels = [["1", "00", 10], ["2", "01", 20], ["3", "10", 30], ["4", "11", 40]]

no_of_legs = [["1", "00", 10], ["2", "01", 20], ["3", "10", 30], ["4", "11", 40]]

speed = [["100", "00", 10], ["200", "01", 20], ["300", "10", 30], ["400", "11", 40]]

weight = [["5", "00", 10], ["10", "01", 20], ["25", "10", 30], ["50", "11", 40]]

material = [["Copper", "00", 10], ["Steel", "01", 20], ["Iron", "10", 30], ["Acrylic", "11", 40]]
```

A.2 Environments

```python
import Data


class Water:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Sand:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Mountain:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Dark:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Obstacles:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Light:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Rubble:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Rocks:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Wind:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height
    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Waterfalls:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Rain:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Glass:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Fire:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Forest:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Collapses:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight


class Temperature:

    def __init__(self):
        self.height = []
        self.width = []
        self.no_of_legs = []
        self.material = []
        self.speed = []
        self.no_of_wheels = []
        self.weight = []

    def get_height(self):
        self.height = [Data.height[0][1], Data.height[3][1]]
        return self.height

    def get_width(self):
        self.width = [Data.width[0][1], Data.width[3][1]]
        return self.width

    def get_legnum(self):
        self.no_of_legs = [Data.no_of_legs[0][1], Data.no_of_legs[3][1]]
        return self.no_of_legs

    def get_material(self):
        self.material = [Data.material[0][1], Data.material[3][1]]
        return self.material

    def get_speed(self):
        self.speed = [Data.speed[0][1], Data.speed[3][1]]
        return self.speed

    def get_wheelnum(self):
        self.no_of_wheels = [Data.no_of_wheels[0][1], Data.no_of_wheels[3][1]]
        return self.no_of_wheels

    def get_weight(self):
        self.weight = [Data.weight[0][1], Data.weight[3][1]]
        return self.weight
```

A.3 Initial Generation

```
import Data
import random


class Initial:

    def __init__(self):
        self.chromosome = ""
        self.cost = 0

    def set_height(self):
        h = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.height[h][1]
        self.cost += Data.height[h][2]

    def set_width(self):
        w = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.width[w][1]
        self.cost += Data.width[w][2]

    def set_speed(self):
        s = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.speed[s][1]
        self.cost += Data.speed[s][2]

    def set_weight(self):
        g = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.weight[g][1]
        self.cost += Data.weight[g][2]

    def set_material(self):
        m = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.material[m][1]
        self.cost += Data.material[m][2]

    def set_wheelsnum(self):
        s = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.no_of_wheels[s][1]
        self.cost += Data.no_of_wheels[s][2]

    def set_legsnum(self):
        l = random.randint(0, 3)
        self.chromosome = self.chromosome + Data.no_of_legs[l][1]
        self.cost += Data.no_of_legs[l][2]

    def set_limbs(self):
        r = random.randint(0, 1024)
        if r % 2 == 0:
            self.set_legsnum()
            self.chromosome += "0"
        else:
            self.set_wheelsnum()
            self.chromosome += "1"

    def create_chromosome(self):
        self.set_height()
        self.set_width()
        self.set_speed()
        self.set_material()
        self.set_weight()
        self.set_limbs()

    def get_chromosome(self):
        return self.chromosome

    def get_cost(self):
        return self.cost
```

A.4 Fitness Function

```
import textwrap


class Fitness:

    def __init__(self, chromosome, condition):
        self.chromosome_fitness = 0
        self.chromosome = chromosome
        self.condition = condition
        self.tokens = textwrap.wrap(self.chromosome, 2)

    def compare_height(self):
        allowed_height = self.condition.get_height()
        for i in range(0, len(allowed_height)):
            if self.tokens[0] == allowed_height[i]:
                self.chromosome_fitness += 1
                break

    def compare_width(self):
        allowed_width = self.condition.get_width()
        for i in range(0, len(allowed_width)):
            if self.tokens[1] == allowed_width[i]:
                self.chromosome_fitness += 1
                break

    def compare_speed(self):
        allowed_speed = self.condition.get_speed()
        for i in range(0, len(allowed_speed)):
            if self.tokens[2] == allowed_speed[i]:
                self.chromosome_fitness += 1
                break

    def compare_material(self):
        allowed_material = self.condition.get_material()
        for i in range(0, len(allowed_material)):
            if self.tokens[3] == allowed_material[i]:
                self.chromosome_fitness += 1
                break

    def compare_weight(self):
        allowed_weight = self.condition.get_weight()
        for i in range(0, len(allowed_weight)):
            if self.tokens[4] == allowed_weight[i]:
                self.chromosome_fitness += 1
                break

    def compare_limbs(self):
        if self.tokens[6] == "0":
            allowed_legs = self.condition.get_legnum()
            for i in range(0, len(allowed_legs)):
                if self.tokens[5] == allowed_legs[i]:
                    self.chromosome_fitness += 1
                    break

        elif self.tokens[6] == "1":
            allowed_wheels = self.condition.get_wheelnum()
            for i in range(0, len(allowed_wheels)):
                if self.tokens[5] == allowed_wheels[i]:
                    self.chromosome_fitness += 1
                    break

    def calculate_fitness(self):
        self.compare_height()
        self.compare_width()
        self.compare_speed()
        self.compare_material()
        self.compare_weight()
        self.compare_limbs()

    def get_fitness(self):
        return self.chromosome_fitness
```

A.5 Crossover

```
import random
import copy


class Crossover:

    # two chromosomes to generate two children
    def __init__(self, chromosomes, crossover_rate):
        self.parent1 = chromosomes[0]
        self.parent2 = chromosomes[1]
        self.crossover_rate = crossover_rate
        self.crossed = []

    def crossover(self):

        # if random.random() <= self.crossover_rate:
        if self.crossover_rate <= 1:
            cross_point = random.randint(1, len(self.parent1) - 1)
            # [:x]-> from the beginning of array to x
            child1 = self.parent1[:cross_point] + self.parent2[cross_point:]
            # [x:]-> from x to the end
            child2 = self.parent2[:cross_point] + self.parent1[cross_point:]

        else:
            # Make copy of parents
            child1 = copy.deepcopy(self.parent1)
            child2 = copy.deepcopy(self.parent2)

        self.crossed = [self.parent1, self.parent2, child1, child2]

    def get_crossed(self):
        return self.crossed
```

A.6 Mutation

```
import random


class Mutation:

    def __init__(self, chromosome):
        # list will contain 4 element from crossover
        self.chromosome = chromosome
        # create list that contain the list from crossover and after mutation  will contain 8 elements
        self.final_chromosome = chromosome

    def mutation(self):
        for i in range(0, 4):
            # convert string to list
            temp_list = list(self.chromosome[i])

            # generate random number
            mutate_bit = random.randint(0, len(self.chromosome[i]) - 1)

            if temp_list[mutate_bit] == "1":
                temp_list[mutate_bit] = "0"

            elif temp_list[mutate_bit] == "0":
                temp_list[mutate_bit] = "1"

            # convert list to string and add to final_chromosome
            self.final_chromosome.append("".join(temp_list))

    def get_chromosomes(self):
        return self.final_chromosome
```

A.7 Algorithm Class

```
import Data
import textwrap
from Initial import Initial
from Fitness import Fitness
from Crossover import Crossover
from Mutation import Mutation


class Algorithm:

    def __init__(self, no_of_chromosomes):
        # The variable for the no_of_chromosomes number for the algorithm to run
        self.no_of_chromosomes = no_of_chromosomes
        # This list will hold the initial generated chromosomes
        self.initials_list = []
        # This list will hold the cost for each initially created chromosome
        self.initials_cost = []
        self.generation_cost = []
        # This list will hold each chromosome and its fitness
        self.fitness_list = [[0]*2 for i in range(0, self.no_of_chromosomes)]
        # This list will hold the most fittest two chromosome
        self.fittest_list = []
        # This list will hold the chromosome sorted according to their fitness
        self.fitness_sorted = []
        # This list will hold the tokens for the fittest chromosome
        self.fittest_tokens_list = []
        # This list will hold the chromosomes after the crossover
        self.crossed_chromosomes = []
        # This list will hold the chromosomes after the mutation
        self.mutated_chromosomes = []
        # This variable is used in the process of calculating the chromosome cost
        self.cost = 0
        # This list will hold the cost of the new chromosomes after mutation
        self.mutated_chromosomes_cost = []

    # Initials generation
    def create_initials(self):
        for i in range(0, self.no_of_chromosomes):
            # Creating object from Initial class
            initial = Initial()
            # Generating an initial chromosome
            initial.create_chromosome()
            # Add the generated chromosome to the initial chromosomes list
            self.initials_list.append(initial.get_chromosome())
            # Add the generated chromosome cost to the initials chromosomes cost list
            self.initials_cost.append(initial.get_cost())

        self.generation_cost = self.initials_cost

    # This function is used to get the initial chromosomes list outside the class
    def get_initials(self):
        return self.initials_list

    # Fitness function
    def calculate_fitness(self, chromosomes_list, condition):
        for i in range(0, self.no_of_chromosomes):
            # Creating object from Fitness class
            fitness = Fitness(chromosomes_list[i], condition)
            # Calculating the fitness of each chromosome
            fitness.calculate_fitness()
            # Adding each chromosome and its fitness to the fitness list
            self.fitness_list[i][0] = chromosomes_list[i]
            self.fitness_list[i][1] = fitness.get_fitness()

        # Sorting the chromosomes according to their fitness
        self.fitness_sorted = sorted(self.fitness_list, key=lambda x: x[1], reverse=True)

        # Creating a list of the most two fittest chromosome
        self.fittest_list = [self.fitness_sorted[0][0], self.fitness_sorted[1][0]]

    # Get the tokens of the best chromosome
    def fittest_tokens_fun(self):
        if self.fitness_sorted[0][1] == 6:
            tokens = textwrap.wrap(self.fitness_sorted[0][0], 2)

            for i in range(0, len(Data.height)):
                if tokens[0] == Data.height[i][1]:
                    self.fittest_tokens_list.append(Data.height[i][0])

            for i in range(0, len(Data.width)):
                if tokens[1] == Data.width[i][1]:
                    self.fittest_tokens_list.append(Data.width[i][0])

            for i in range(0, len(Data.speed)):
                if tokens[2] == Data.speed[i][1]:
                    self.fittest_tokens_list.append(Data.speed[i][0])

            for i in range(0, len(Data.material)):
                if tokens[3] == Data.material[i][1]:
                    self.fittest_tokens_list.append(Data.material[i][0])

            for i in range(0, len(Data.weight)):
                if tokens[4] == Data.weight[i][1]:
                    self.fittest_tokens_list.append(Data.weight[i][0])

            if tokens[6] == "0":
                for i in range(0, len(Data.no_of_legs)):
                    if tokens[5] == Data.no_of_legs[i][1]:
                        self.fittest_tokens_list.append(Data.no_of_legs[i][0])

                        # to check  that there is legnum
                        self.fittest_tokens_list.append("0")
            elif tokens[6] == "1":
                for i in range(0, len(Data.no_of_wheels)):
                    if tokens[5] == Data.no_of_wheels[i][1]:
                        self.fittest_tokens_list.append(Data.no_of_wheels[i][0])

                        # to check  that there is wheelnum
                        self.fittest_tokens_list.append("1")
    def get_fittest_tokens(self):
        return self.fittest_tokens_list

    # Crossover
    def do_crossover(self):
        crossover = Crossover(self.fittest_list, 1)
        crossover.crossover()
        self.crossed_chromosomes = crossover.get_crossed()

    # Mutation
    def do_mutation(self):
        mutation = Mutation(self.crossed_chromosomes)
        mutation.mutation()
        self.mutated_chromosomes = mutation.get_chromosomes()

        # Calculating the cost of each mutated chromosome
        self.mutated_chromosomes_cost = []
        for c in range(0, len(self.mutated_chromosomes)):
            tokens = textwrap.wrap(self.mutated_chromosomes[c], 2)

            self.cost = 0
            for i in range(0, len(Data.height)):
                if tokens[0] == Data.height[i][1]:
                    self.cost += Data.height[i][2]

            for i in range(0, len(Data.width)):
                if tokens[1] == Data.width[i][1]:
                    self.cost += Data.width[i][2]

            for i in range(0, len(Data.speed)):
                if tokens[2] == Data.speed[i][1]:
                    self.cost += Data.speed[i][2]

            for i in range(0, len(Data.material)):
                if tokens[3] == Data.material[i][1]:
                    self.cost += Data.material[i][2]

            for i in range(0, len(Data.weight)):
                if tokens[4] == Data.weight[i][1]:
                    self.cost += Data.weight[i][2]

            if tokens[6] == "0":
                for i in range(0, len(Data.no_of_legs)):
                    if tokens[5] == Data.no_of_legs[i][1]:
                        self.cost += Data.no_of_legs[i][2]
            elif tokens[6] == "1":
                for i in range(0, len(Data.no_of_wheels)):
                    if tokens[5] == Data.no_of_wheels[i][1]:
                        self.cost += Data.no_of_wheels[i][2]

            self.mutated_chromosomes_cost.append(self.cost)

        self.generation_cost = self.mutated_chromosomes_cost

    def get_mutated(self):
        return self.mutated_chromosomes

    def get_generation_cost(self):
        return self.generation_cost

    def the_most_fittest_chromosome(self):
        return self.fitness_sorted[0][1]
```

A.8 GUI

```
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys


Gui_app = QApplication(sys.argv)    # create application
main_gui = QWidget()                 # main window


main_gui.setWindowIcon(QIcon('img/robotic2.jpg'))   # set icon

main_gui.setWindowTitle("Robot Emulator")            # set title

# The background set
bc = QPalette()
main_gui.setPalette(bc)


# set window size
main_gui.resize(1200,700)   # set size
main_gui.move(500,200)      # set position

# set choose label
choose_label = QLabel(main_gui)                             # Create label
```

```
choose_label.setText("<h1> Choose The Environment </h1> ")
choose_label.move(750, 20)                                  # Set its position

# radio boxes
water_radio = QRadioButton(main_gui, text="Water")
water_radio.move(750,90)

sand_radio = QRadioButton(main_gui, text="Sand")
sand_radio.move(1000,90)

dark_radio = QRadioButton(main_gui, text="Dark")
dark_radio.move(750,140)

mountain_radio = QRadioButton(main_gui, text="Mountain")
mountain_radio.move(1000,140)

light_radio = QRadioButton(main_gui, text="Light")
light_radio.move(750,190)

obstacles_radio = QRadioButton(main_gui, text="Obstacles")
obstacles_radio.move(1000,190)

rocks_radio = QRadioButton(main_gui, text="Rocks")
rocks_radio.move(750,240)

rubble_radio = QRadioButton(main_gui, text="Rubble")
rubble_radio.move(1000,240)

wind_radio = QRadioButton(main_gui, text="Wind")
wind_radio.move(750,290)

waterfalls_radio = QRadioButton(main_gui, text="Waterfalls")
waterfalls_radio.move(1000,290)

rain_radio = QRadioButton(main_gui, text="Rain")
rain_radio.move(750,340)

collapses_radio = QRadioButton(main_gui, text="Collapses")
collapses_radio.move(1000,340)

glass_radio = QRadioButton(main_gui, text="Glass")
glass_radio.move(750,390)

temperature_radio = QRadioButton(main_gui, text="Temperature")
temperature_radio.move(1000,390)

fire_radio = QRadioButton(main_gui, text="Fire")
fire_radio.move(750,440)

forest_radio = QRadioButton(main_gui, text="Forest")
forest_radio.move(1000,440)

# set result label
result_label = QLabel(main_gui)
result_label.setText("<h1><b> The robot Characteristics  <b></h1> ")
result_label.move(10, 20)


height_label = QLabel(main_gui)
height_label.setText("<h3>  Height <h3> ")
height_label.move(20,100)

height_result = QLineEdit(main_gui)
height_result.move(200,90)
height_result.resize(120,40)

width_label = QLabel(main_gui)
width_label.setText("<h3> Width <h3> ")
width_label.move(20,170)

width_result = QLineEdit(main_gui)
width_result.move(200,160)
width_result.resize(120,40)

speed_label=QLabel(main_gui)
speed_label.setText("<h3> Speed <h3> ")
speed_label.move(20,240)

speed_result = QLineEdit(main_gui)
speed_result.move(200,230)
speed_result.resize(120,40)

weight_label = QLabel(main_gui)
weight_label.setText("<h3>  Weight <h3> ")
weight_label.move(20,310)

weight_result = QLineEdit(main_gui)
weight_result.move(200,300)
weight_result.resize(120,40)

material_label = QLabel(main_gui)
```

```
material_label.setText("<h3>  Material <h3> ")
material_label.move(20,380)

material_result = QLineEdit(main_gui)
material_result.move(200,370)
material_result.resize(120,40)

wheelnum_label = QLabel(main_gui)
wheelnum_label.setText("<h3>  No of wheels  <h3> ")
wheelnum_label.move(20,450)

wheelnum_result = QLineEdit(main_gui)
wheelnum_result.move(200,440)
wheelnum_result.resize(120,40)

legnum_label = QLabel(main_gui)
legnum_label.setText("<h3>  No of legs <h3> ")
legnum_label.move(20,520)

legnum_result = QLineEdit(main_gui)
legnum_result.move(200,510)
legnum_result.resize(120,40)

cost_lable = QLabel(main_gui, text=" <h3> Implementation Cost<h3>")
cost_lable.move(20,590)

cost_result = QLineEdit(main_gui)
cost_result.move(200,580)
cost_result.resize(120,40)

time_label = QLabel(main_gui)
time_label.setText("<h4>  Time of Execution <h4> ")
time_label.move(460,555)

time_result = QLineEdit(main_gui)
time_result.move(450,580)
time_result.resize(150,40)

gen_num_label = QLabel(main_gui)
gen_num_label.setText("<h4>    No. of Generations  <h4> ")
gen_num_label.move(460, 460)

gen_num_result = QLineEdit(main_gui)
gen_num_result.move(450, 485)
gen_num_result.resize(150, 40)

# GUI buttons
start_btn = QPushButton("Start", main_gui)
start_btn.setGeometry(700, 550, 100, 50)
start_btn.setToolTip("click to run")

draw_btn = QPushButton("Draw", main_gui)
draw_btn.setGeometry(900, 550, 100, 50)
draw_btn.setToolTip("click to draw")

# Exit button
exit_btn = QPushButton("Exit", main_gui)
exit_btn.setGeometry(1100, 550, 100, 50)
exit_btn.clicked.connect(exit)
exit_btn.setToolTip("click to exit")
```

A.9 GUI Data Filling

```
from Gui import *
from Conditions import *
from Algorithm import Algorithm
import time

import matplotlib.pyplot as plt


# choose the condition function
def condition():

    if water_radio.isChecked():
        return Water()

    elif sand_radio.isChecked():
        return Sand()

    elif dark_radio.isChecked():
        return Dark()

    elif mountain_radio.isChecked():
        return Mountain()

    elif light_radio.isChecked():
        return Light()
    elif obstacles_radio.isChecked():
        return Obstacles()

    elif rocks_radio.isChecked():
        return Rocks()

    elif rubble_radio.isChecked():
        return Rubble()

    elif wind_radio.isChecked():
        return Wind()

    elif waterfalls_radio.isChecked():
        return Waterfalls()

    elif rain_radio.isChecked():
        return Rain()

    elif collapses_radio.isChecked():
        return Collapses()

    elif glass_radio.isChecked():
        return Glass()

    elif temperature_radio.isChecked():
        return Temperature()

    elif fire_radio.isChecked():
        return Fire()

    elif forest_radio.isChecked():
        return Forest()


def print_characteristics(chromosome_list):
    height_result.setText("          " + chromosome_list[0] + " cm")
    width_result.setText("          " + chromosome_list[1] + " cm")
    speed_result.setText("          " + chromosome_list[2] + " m/s")
    material_result.setText("       " + chromosome_list[3])
    weight_result.setText("           " + chromosome_list[4] + " kg")

    if chromosome_list[6] == "0":
        legnum_result.setText("           " + chromosome_list[5])
        wheelnum_result.setText("           -")

    elif chromosome_list[6] == "1":
        wheelnum_result.setText("           " + chromosome_list[5])
        legnum_result.setText("           -")


def calculate_cost(l):

    cost = 0

    for i in range(0, len(Data.height)):
        if l[0] == Data.height[i][0]:
            cost += Data.height[i][2]

    for i in range(0, len(Data.width)):
        if l[1] == Data.width[i][0]:
            cost += Data.width[i][2]

    for i in range(0, len(Data.speed)):
        if l[2] == Data.speed[i][0]:
            cost += Data.speed[i][2]

    for i in range(0, len(Data.material)):
        if l[3] == Data.material[i][0]:
            cost += Data.material[i][2]

    for i in range(0, len(Data.weight)):
        if l[4] == Data.weight[i][0]:
            cost += Data.weight[i][2]

    if l[6] == "0":
        for i in range(0, len(Data.no_of_legs)):
            if l[5] == Data.no_of_legs[i][0]:
                cost += Data.no_of_legs[i][2]

    if l[6] == "1":
        for i in range(0, len(Data.no_of_wheels)):
            if l[5] == Data.no_of_wheels[i][0]:
                cost += Data.no_of_wheels[i][2]

    return cost


def draw(lx, y):

    x = []
    for i in range(1, lx + 1):
        x.append(i)

    plt.title("Robot Emulator")
    plt.xlabel("Generations")
    plt.ylabel("max fitness")
    plt.axis([1, 9, 0, 7])

    plt.plot(x, y)


def draw_show():
    plt.show()
```

A.10 Main application

```
from GuiData import *


def main_fun(no_of_iterations):

    # Start time of execution
    start = time.time()

    # We specify the number of initially generated chromosomes
    main = Algorithm(8)

    selected_condition = condition()

    main.create_initials()

    chromosomes_list = main.get_initials()

    # List for the most fittest chromosomes in each generation
    fittest_list_to_draw = []

    for i in range(0, no_of_iterations):

        main.calculate_fitness(chromosomes_list, selected_condition)

        f = main.the_most_fittest_chromosome()
        fittest_list_to_draw.append(f)

        if (main.fitness_sorted[0][1] == 6) or (i == (no_of_iterations - 1)):

            # calculate the characteristics
            main.fittest_tokens_fun()
            # list with the characteristics
            fittest_list = main.get_fittest_tokens()

            print_characteristics(fittest_list)

            # calculate the cost
            cost = calculate_cost(fittest_list)
            cost_result.setText("         " + str(cost)+" $")

            draw(i + 1, fittest_list_to_draw)

            # Time of execution
            t = (time.time() - start) * 1000
            time_result.setText("            " + str(int(t))+" ms")

            gen_num_result.setText("              " + str(i + 1))
            break

        main.do_crossover()
        main.do_mutation()
        chromosomes_list = main.get_mutated()


start_btn.clicked.connect(lambda: main_fun(10))

draw_btn.clicked.connect(draw_show)

# start GUI
main_gui.show()
Gui_app.exec_()
```