Project Report

for

Team 3.14

(Team π)

Members

Chung Yin Leung
Erik Xu
Nathan Wong

I. Project's Program Summary

Our program uses a preset database or a custom database that the user may load. The program uses the area data, solar insolation data, and wind speed data of a city at a particular month to determine a solar farm and or a wind farm whose area together covers 0.0001% of the area of the city. The user inputs the percentage of the 0.0001% area that will be allocated to the solar farm and wind farm. The solar farm uses SunPower 425-watt solar panels[1] while the wind farm uses Enercon E-126 wind turbines[2]. With the specifications of those solar panels and wind turbines, the program calculates the power output of each farm using the data from the database and estimates the number of electric vehicles that can be supported. It is necessary to note that the electric vehicles are using the specifications of the Nissan Leaf[3].

Using this estimate, our program starts a simulation in which the estimated number of electric vehicles randomly moves between places where electric vehicles can park. Each place has a maximum amount of space for electric vehicles and a percentage that states how much electric vehicles will be charging from the power grid. This percentage and the amount of power produced by the power grid are affected by time. The user may add more places for the electric vehicles to park and or add more electric vehicles to the simulation. If the total power demanded by the electric vehicles are within 10% of the total power produced by the power grid, the scale of the farms will increase by devoting 0.00005% of the land area of the city to the farms. The simulation continues until the user wants to end it.

Assumptions Made

- We use 0.0001% of the land for energy producers at the beginning. Then 0.00005% more land will be allocated as needed. We're using a small percentage because greater than 0.0001% causes too much EVs to be supported and our computers cannot handle that many objects.
- The month is static; it will not change after the program starts.
- All the EVs are charging no matter where they go because the power demanded by the EVs are

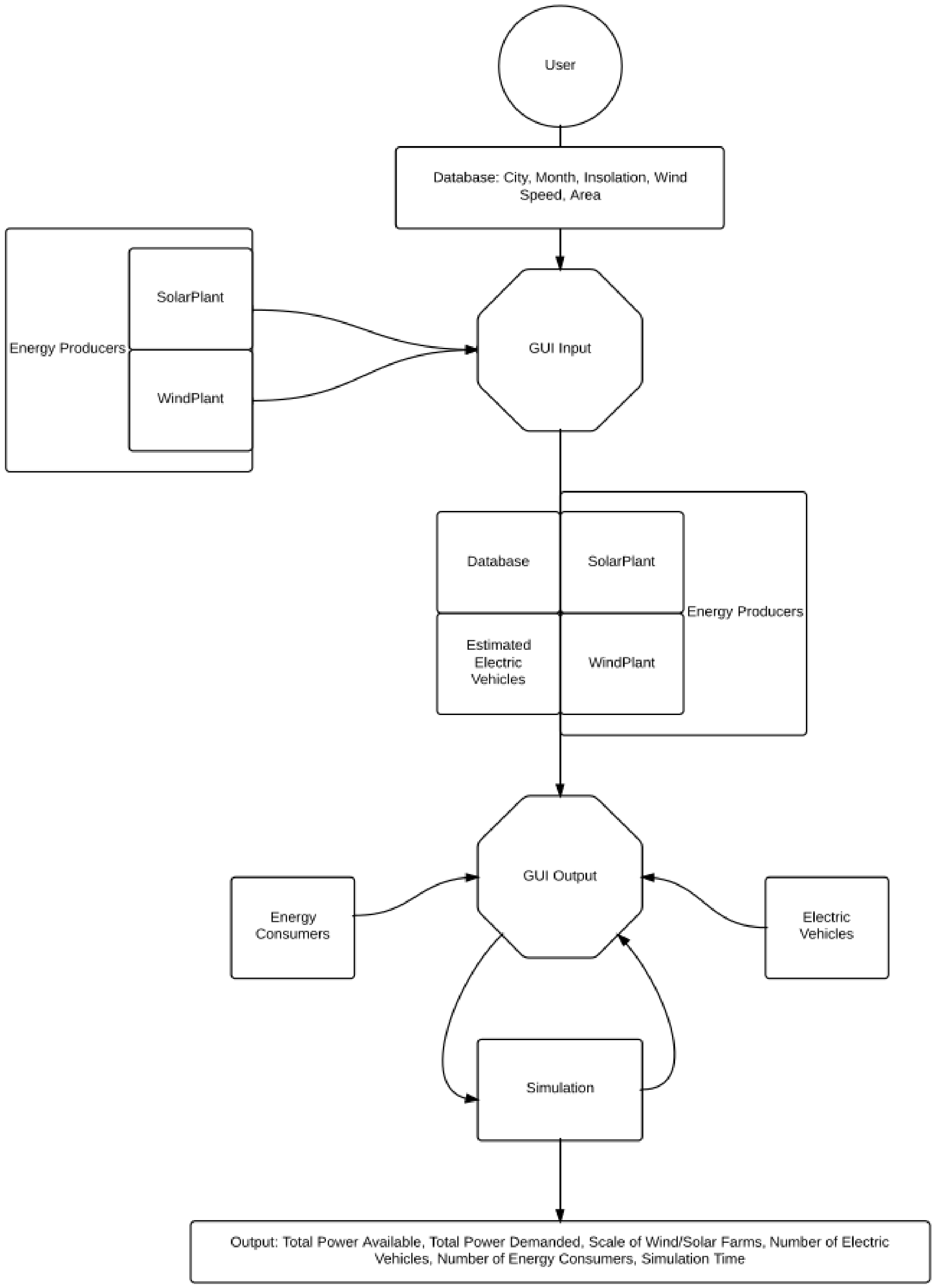
[1] http://us.sunpowercorp.com/cs/Satellite?blobcol=urldata&blobheadername1=Content-Type&blobheadername2=Content-Disposition&blobheadervalue1=application%2Fpdf&blobheadervalue2=inline%3B+filename%3Dsp_425Ewh_en_ltr_ds_p_shortercable.pdf&blobkey=id&blobtable=MungoBlobs&blobwhere=1300276455310&ssbinary=true

[2] http://www.enercon.de/p/downloads/ENERCON_PU_en.pdf

[3] http://www.nissanusa.com/electric-cars/leaf/charging-range/charging/

too low and implementing different amounts of charging EVs will slow down execution time dramatically with our current setup. We did not have time to change our approach.
- The power produced at different times are not based on average yearly values. Instead, the power produced is based on the power output of the Wind and Solar plants modified by the AYV percentage modifiers.

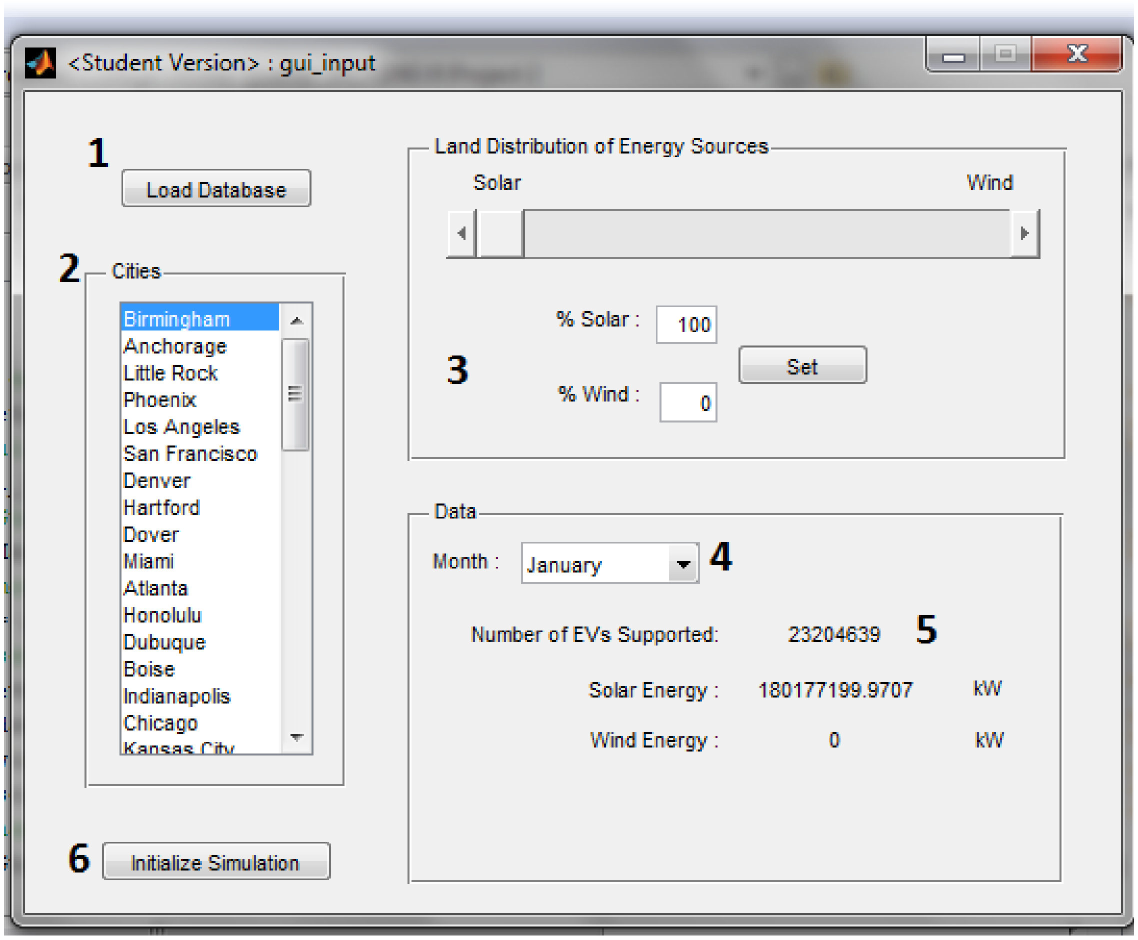
II. Layout of the Program


User
Database: City, Month, Insolation, Wind Speed, Area
Energy Producers
SolarPlant
WindPlant
GUI Input
Database
SolarPlant
Energy Producers
Estimated Electric Vehicles
WindPlant
GUI Output
Energy Consumers
Electric Vehicles
Simulation
Output: Total Power Available, Total Power Demanded, Scale of Wind/Solar Farms, Number of Electric Vehicles, Number of Energy Consumers, Simulation Time

III. GUI Design - Input

The input GUI basically receives a database from the user or uses the preloaded one from Project 1. The GUI then uses the classes of the Energy Producers (WindPlant and SolarPlant) are used to determine the number of electric vehicles that can be supported by finding the total power produced when only 10% of the area of the city is distributed to the Energy Producers. This area is further divided between wind energy and solar energy through user input. The classes of the Energy Producers will be explained further on page 7.

Next is an examination of the GUI.



% Solar : 100

% Wind : 0

Month : January

Number of EVs Supported: 23204639

Solar Energy : 180177199.9707 kW

Wind Energy : 0 kW

When the program is run, the GUI, as shown above, will be displayed. (The real interface does not have the bolded numbers.):

1. The load database button allows the user to load a custom database into the program and

replace the default database that is preloaded.

2. The cities listbox allows the user to select a city from the database.
3. Next, the user is allowed to determine the how a certain amount of land in the city will be distributed amongst a solar farm and a wind farm. The user may use a slider or the edit boxes to set the percentages. If the user edits the percentages boxes, he or she must click on the 'Set' button. This will check whether the percentages are valid. If they are invalid, an error message will display and explain why.
4. The user may select a specific month's insolation and wind speed data to be used.
5. Data on the estimated number of electric vehicles (EV) supported, the solar energy produced, and the wind energy produced are displayed here.
6. The 'Initialize Simulation' button begins the simulation. Note that only one simulation window can be opened at one time.

Notes on the Database

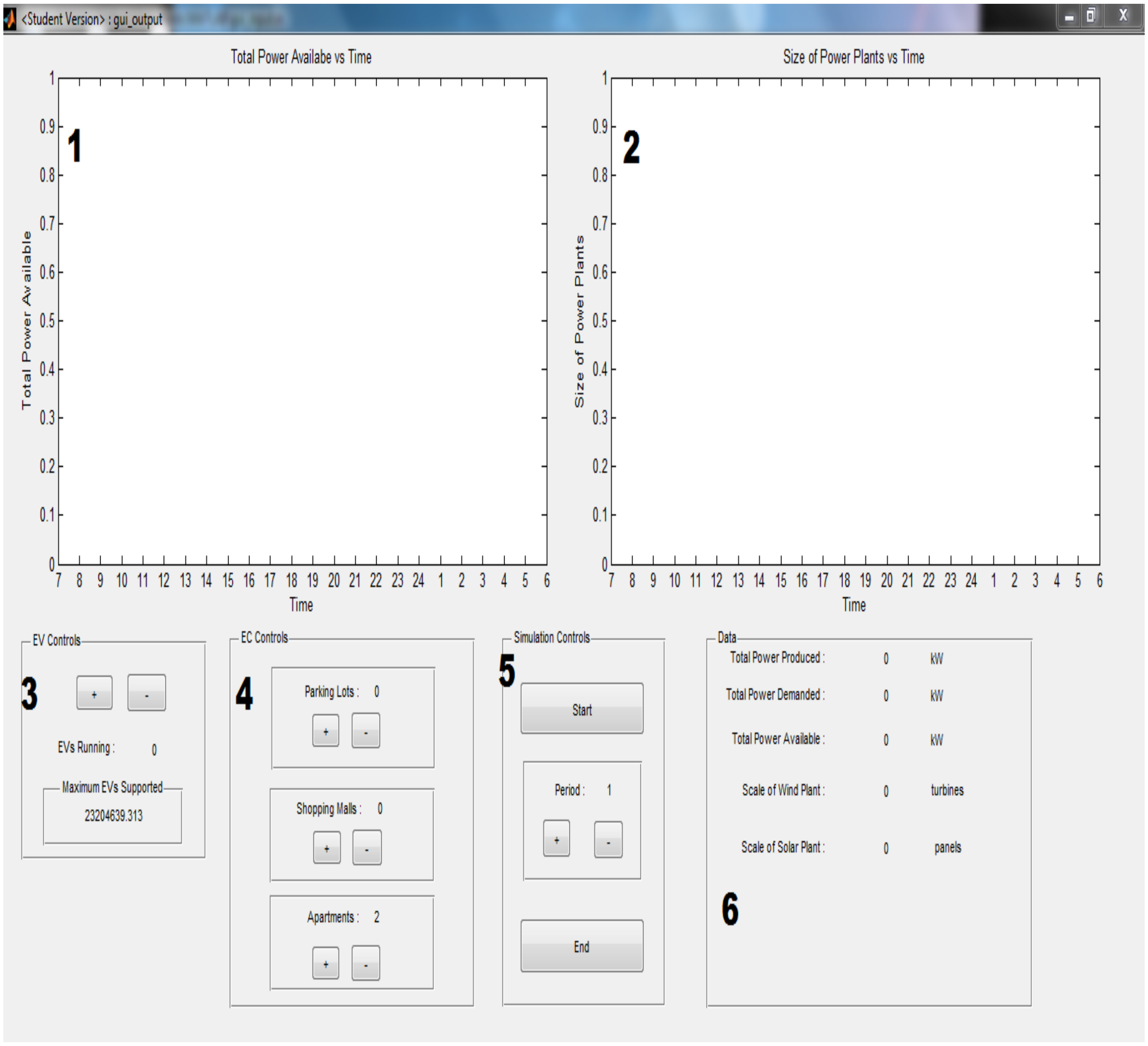
A valid custom database is one that has seven fields with the following as names, shown exactly below:

- **area**
- **city**
- **population**
- **precip**
- **solar**
- **state**
- **wind**

It does not have to be in that order; there must be fields that are named as such, case sensitive.

And for best results, the row of each field's data should correspond to the data's city in the **city** field.

IV. GUI Design - Output



EVs Running : 0

Maximum EVs Supported: 23204639.313

Parking Lots : 0

Shopping Malls : 0

Apartments : 2

Period : 1

Total Power Produced : 0 kW

Total Power Demanded : 0 kW

Total Power Available : 0 kW

Scale of Wind Plant : 0 turbines

Scale of Solar Plant : 0 panels

The output GUI pops up after the 'Initialize Simulation' button is pressed in the input GUI.
1. Graph of the Total Power Available vs Time.

2. Graph of the Size of Power Plants vs Time.
3. Option to add or subtract the number of EVs. If the number of EV’s is 0 then it will display, “You have no more EVs to remove.”
4. Option to add or subtract the number of Energy Consumers: Parking Lots, Shopping Malls, and Apartments. If the number of Energy Consumers is 2 then it will display, “You cannot have less than two energy consumers.”
5. The user is allowed to choose when to start, pause, and end the simulation.
6. Shows the data that is made during the simulation. Shows Total Power Produced, Total Power Demanded, Total Power Available, Scale of Wind Plant, and Scale of Solar Plant.

Notes on the GUIs

The memory handles of both GUIs are stored through setappdata. The input GUI is stored in the root 0 while the output GUI is stored in the memory handle of the input GUI. This allows both GUI to access each other GUI’s guidata (handles structure).

V. Energy Producers

The Energy Producers consist of two classes: SolarPlant and WindPlant. Both classes have nearly exact properties and functions.

The SolarPlant needs to be constructed using a scalar insolation data for a month and the area allocated to solar energy. It first finds the number of solar panels that can be built in that area following the specification of the SunPower 425-watt solar panels in which each panel is 2.16 m^2. This is simply the area divided by the size of each panel and then rounded down. Then it calculates the total power produced using the number of solar panels, the panels' efficiency, which we rounded to 20%, and the insolation data.

Aside from that, the SolarPlant class has functions to return the power produced and the number of panels, and functions to update the number of panels and the insolation data. Updating the properties of the SolarPlant will also automatically update the power produced. Lastly, it has a function to output a time modified power output by taking in time as an argument.

Source:
http://us.sunpowercorp.com/cs/Satellite?blobcol=urldata&blobheadername1=Content-Type&blobheadername2=Content-Disposition&blobheadervalue1=application%2Fpdf&blobheadervalue2=inline%3B+filename%3Dsp_425Ewh_en_ltr_ds_p_shortercable.pdf&blobkey=id&blobtable=MungoBlobs&blobwhere=1300276455310&ssbinary=true

The WindPlant is similar. It needs to be constructed using a scalar wind speed data for a month and the area allocated to wind energy. It first stores the wind speed. Then it finds the number of wind turbines, which is just the area divided by the size of each turbine, 12,668 m^2, and rounded down. The size does not accurately represent the area that a wind turbine takes up in real life because that value is just the area that the turbine's fans swept through.

Continuing on, the power is found by using a separate function called calc_wind. It takes a wind value as an input variable and sorts the wind speed to match a power output so that the correct power output at that wind speed will be returned. This function is to save space in the class.With the estimated power output at a wind speed, the total power produced is simply that estimated power output multiplied with the number of wind turbines.

The WindPlant class has almost the same functions as the SolarPlant class. It has functions to return the power produced and the number of wind turbines, and functions to update the number of wind turbines and the wind speed the plant experiences. It also has a function to output a time modified power output by taking in time as an argument.

Source: http://www.enercon.de/p/downloads/ENERCON_PU_en.pdf

VI. Energy Consumers

The Energy Consumers are represented by the class, Energyconsumers.
The class has five properties: consumer_type, num_cars, and total_space and the constant properties: type and ev_chrg. ‘consumer_type’ is the property used to indicate whether it’s a parking lot, shopping mall or an apartment. ‘num_cars’ is the amount of electric vehicles that are currently at the energy consumer location. ‘total_space’ is the number of electric vehicles spaces dedicated for each of the energy consumers. The constant property ‘type’ is used to randomly assign the energy consumer a type when an object is created and the ‘ev_chrg’ is the number of EVs charging at certain times of the day. The clas has a function called ‘checkFull()’ which determines whether energyconsumer has any empty EV spaces for charging.

VII. Electric Vehicles

Electric Vehicles consists of one class called EV. There is one variable named ‘con.’ When an EV is created, it takes an array of Energyconsumers as arguments and makes it equal to ‘con’. If there is no arguement or more than one, then ‘con’ is just a Energyconsumer object. In the move function, it finds the length of this array given and makes a randomized vector of the length that does not repeat by using randperm(). By looking at randomly indices of the array of Energyconsumers, it checks the Energyconsumer function checkFull() to see if there is any room. If that Energyconsumer is not full, then the Energyconsumer variable num_car will have one added to it. If it is full, then move() will keep looping until there is an empty one.

Appendix A

Yout

Table of Tasks

Tasks:	**Description:**	**Responsible Team Members:**
Energy Producers	Research, design ,and code the energy producers for the simulation	Erik Xu
Energy Consumers	Design and code the energy consumers for the simulation	Nathan Wong, Erik Xu
Electric Vehicles	Design and code a class to represent Electric Vehicles	Nathan Wong
GUI Design - Input	Design and code the input part of the GUI	Chung Yin Leung
GUI Design - Output	Design and code the output part of the GUI	Chung Yin Leung, Erik Xu

Contribution Summaries

Erik Xu

I first researched about the solar panels and wind turbines of the top energy producers in the world. After, we selected the solar panel and wind turbine to use and I created the 'Energyconsumers' class for the parking lot, shopping mall and apartment which has a type and total space input argument for its constructor. In addition, it has two functions, 'checkfull()' to check if the number of cars is equal to the total spaces and 'changeEV()' to change the maximum amount of EV spaces charging during certain times of the day. In type, 1 corresponds to a Parking Lot, 2 corresponds to a Shopping Mall, and 3 corresponds to an Apartment object. If there there is more or less than one argument, then Energyconsumer will randomly be one of the three options. Besides that, I also helped work on the GUI output to implement the functions for the buttons to increase the amount of energy consumers and electric vehicles. Lastly, I helped make the data table display the correct results based on our calculations.

Nathan Wong

I started helped start the input gui ideas like the slider but it was changed later. We researched the various solar panels and wind turbines for our project. I created the EV class which makes Electric Vehicle objects. It takes Energyconsumer object arrays as arguments and randomly picks one to add one to the num_car variable. If it can it will, if not it will randomly pick the next Energyconsumer to try to add. It is needed to help fill up the Energy Consumers. I tried to help out with the gui output when I was done with the EV class.

Chung Yin Leung

I researched and found the types of solar panels and wind turbines that we used for our project. They're not the best, but both types have data that we can easily find and use. I designed and coded the input part of the GUI. It can load a database file so that users are not limited to one database, and the loaded cities are displayed so the user can pick which city he or she wants. I coded that and pretty much every little details in the GUI. I've also coded the calculations that are done to display the data in the input GUI and ways to pass on data to the output GUI. But I also designed the output part of the GUI and helped my group member in implementing plots and the Timer object for the simulation.

All team members have read the task summaries contained in this report and have been given an opportunity to comment.