



Project Report

Project Option #1: Sustainable Urban Planning

ENG6, Prof. Amirtharajah, Winter 2013

teamName = []

Jiun Rong Chen
Wesley Ho
Nicholas Yee-Toy

Tables of Contents

1. Overview	3
2. Functionalities of Each File	
weather.mat	3
pricing.mat, map.mat	4
grid.mat, database.m	5
city.m	6
GUI.m	7
GUI.fig	8
ion.jpg	9
3. How the Program works	9
4. Problems Encountered & Solutions	10
5. Appendix	11

1. Overview

This project requires us to write a program that allows the user to deploy solar and wind energy for a small area, given a list of solar panels and wind turbines. We've selected and included the maps of San Francisco, Los Angeles, and Las Vegas for the user to deploy the reusable energies.

We will approach this project using a top-down approach by implementing object-oriented capabilities of MATLAB. The backbone of the project will be structured into two classes, the city class and the database class. The city class will contain solar and wind data unique to each of the three cities and methods that allow access to this data. The database class will contain the database from Project 1 and methods that can retrieve this data. The majority of the coding and implementation is contained in the classes while the GUI requires minimal coding, since it only retrieves and displays the data.

The breakdown of the functionalities of each file is explained in detail in the next section. For the overall relationship between each file, go to "3. How the Program Works". The YouTube link is: <http://www.youtube.com/watch?v=o1z8ih5lyGw>

2. Functionalities of Each File

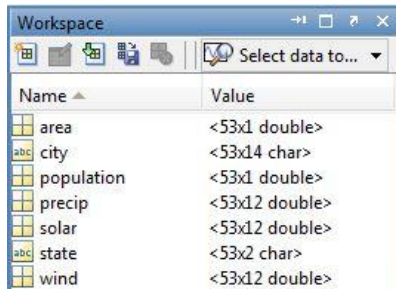
This program consists of the following files:

weather.mat
pricing.mat
map.mat
grid.mat
database.m
city.m
GUI.m
GUI.fig
ion.jpg

The functionalities of each file are the following:

File **weather.mat**

This is exactly the same file given from Project 1. Although we don't need all of the data, we leave it as part of the file and retrieve the specific arrays (solar and wind data) with a different m-file.

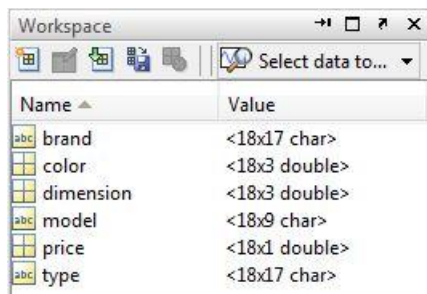


The image shows a MATLAB Workspace window with a table of variables. The variables are area, city, population, precip, solar, state, and wind. Each variable has a corresponding data type indicated in the 'Value' column.

Name	Value
area	<53x1 double>
city	<53x14 char>
population	<53x1 double>
precip	<53x12 double>
solar	<53x12 double>
state	<53x2 char>
wind	<53x12 double>

File **pricing.mat**

It stores arrays of all the information relating to the solar panels and wind turbines. The brand, model, and type data are stored as arrays of strings; while the dimension, price, and color data are stored as arrays of numbers.



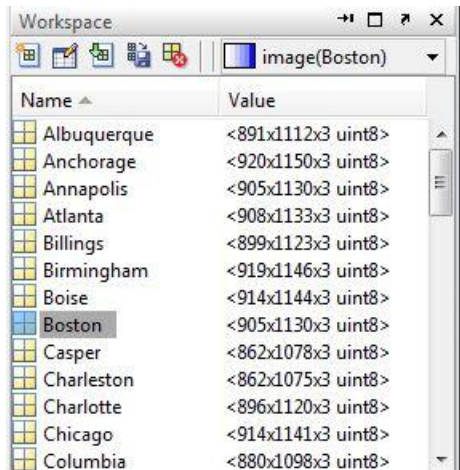
The image shows a MATLAB Workspace window with a table of variables. The variables are brand, color, dimension, model, price, and type. Each variable has a corresponding data type indicated in the 'Value' column.

Name	Value
brand	<18x17 char>
color	<18x3 double>
dimension	<18x3 double>
model	<18x9 char>
price	<18x1 double>
type	<18x17 char>

All of the data are given in the assignment PDF, except the color array. The color is going to be used to differentiate different turbine/panel selected in the GUI, and we have chosen the color of our choice. The 3 columns of numbers are the RGB colors, which MATLAB recognizes.

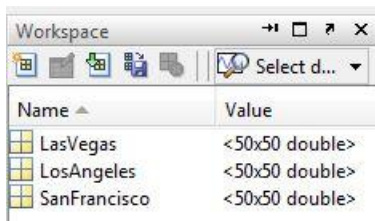
File **map.mat**

It stores arrays of map image data for all the cities. The Google images for each city has been processed in MATLAB by "imread()" function, and the arrays are saved as one mat-file. Upon calling for the image, the array of the specified city will be called, instead of png or jpg files. Due to the high resolutions of each image, the size of each image is quite large and the amount of them contributes to most of the file size.



File **grid.mat**

It contains 50 by 50 matrices for each city, with each index corresponding to the type of the solar panels or wind turbines that the user places on the grid. This means that our grid contains a total of 50×50 , or 2500 boxes to click from. If no panel or turbine has been placed, the default number for that grid box will be 0; otherwise the number represents a panel or turbine.



File **database.m**

The data that the GUI will need are stored as objects by object-oriented programming. The variables are below:

```

>> database

ans =

    database handle

Properties:
    city: [53x14 char]
    solar: [53x12 double]
    wind: [53x12 double]
    map: [1x1 struct]
    brand: [18x17 char]
    dimension: [18x3 double]
    model: [18x9 char]
    price: [18x1 double]
    type: [18x17 char]
    color: [18x3 double]
    grid: [1x1 struct]

    Methods, Events, Superclasses

```

The constructor loads the data from all the previously described mat-files.

The sources of the variables are:

weather.mat (city, solar, and wind variables)

map.mat (map variable)

pricing.mat (brand, dimension, model, price, type, color, grid variables)

Functions within database.m are below:

returnCity(obj)	: returns list of cities as cell arrays
returnSolar(obj, city)	: returns the average solar data of the specified city over 12 months
returnWind(obj, city)	: returns the average wind data of the specified city over 12 months
returnMap(obj, city)	: returns the map data of specified city
returnModel(obj)	: returns the list of models given from the assignment PDF
returnNumber(obj)	: returns the total number of solar panels available
returnBrand(obj, model)	: returns the name of brand of the specified model as cell arrays
returnDimension(obj, model)	: returns the dimensions of the specified model as cell arrays
returnPrice(obj, model)	: returns the price of the specified model
returnType(obj, model)	: returns the type of the specified model
returnColor(obj, model)	: returns the grid RGB color of the specified model
returnGrid(obj, name)	: returns the grid of the specified city
setGrid(obj, grid, city)	: saves the grid of the specified city into grid.mat

File **city.m**

It stores the grid and the bill of materials data specific to each city. It retrieves basic data from database.m and further changes and calculations are done according to user input in the GUI.

city handle

```
Properties:
  name: 'SanFrancisco'
  grid: [50x50 double]
  BOM: []
  dbase: [1x1 database]
  dimBox: 35
  dimMap: [50 50]
  mph2Kw: 500
```

Methods, Events, Superclasses

The above screenshot is an example of just the object for "San Francisco", and the variable format is the same for the other two cities. Unlike database.m, a string variable is required for the constructor and some properties are actually constants. The source of each property is explained below:

name: obtained when constructed
grid: obtained directly from database.m
BOM: set by the function "createBOM"
dbase: the database.m

dimBox, dimMap, and mph2Kw are constants, and they're set as:

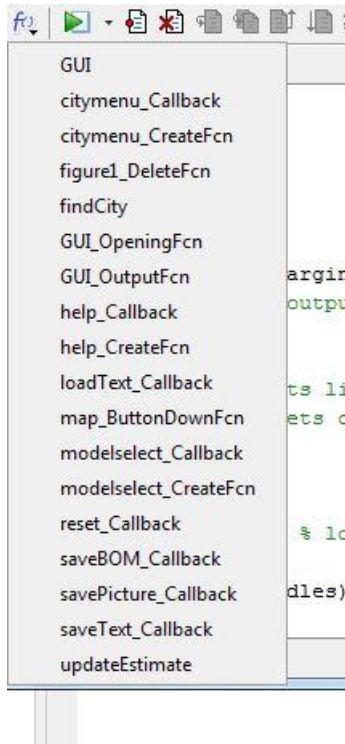
dimBox: 35ft for width and length
dimMap: 50x50 squares in the grid
mph2Kw: 500 kW / 1mph wind, which was given from the assignment PDF

Functions within class.m are below:

returnDimMap(obj)	: returns the dimensions of the map/grid, which set to be 50x50
returnName(obj)	: returns the name of city as a string
returnMap(obj)	: uses "returnMap" function from database.m and returns the city map data
createBOM(obj)	: creates a chart for the bill of materials that the user has used, and stores the detailed information in BOM
returnBOM(obj)	: simply returns BOM variable
estimateSolar(obj)	: using average solar data, estimates the solar energy provided by the solar panels the user builds
estimateWind(obj)	: using average wind data, estimates the wind energy provided by the wind turbines or generators the user builds
outputGrid(obj)	: saves the grid changes by the user back into database.m
updateGrid(obj, x, y, model)	: changes the grid number of specific index (x,y) to the corresponding model location number, which is then used to retrieve the RGB color
returnGrid(obj)	: simply returns the grid matrix
loadGrid(obj, grid)	: saves grid in both class.m and database.m

File GUI.m

This file is the graphical user interface, with structures and formats designed by MATLAB.



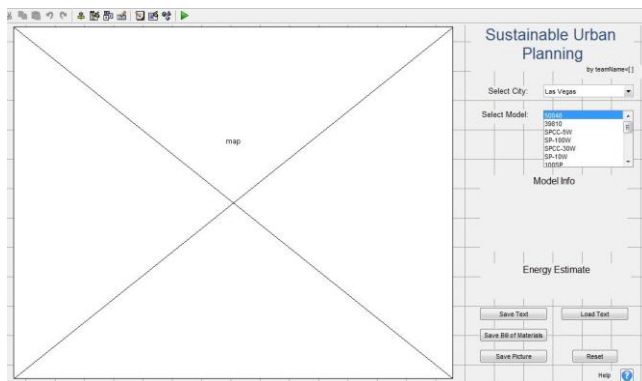
The specific functions are below:

- GUI_OpeningFcn : runs upon startup of the GUI; retrieves data from database.m and calls the functions necessary to display the city data (such as the map and the grid) and the model data; it also displays the instructions in a message box
- figure1_DeleteFcn : upon closing of the GUI, class.m function "outputGrid" is called and saves the grid data after the user input
- GUI_OutputFcn : returns the output to the command line, which we did not use
- citymenu_CreateFcn : creates a list of city names to be displayed at the city menu
- citymenu_Callback : updates the grid and energy estimate when the user selects a different city from the menu
- map_ButtonDownFcn : changes the grid color corresponding to the user's choice of model when left-clicked on map: clears the color of the specified grid when right-clicked
- modelselect_CreateFcn: creates a list of models to be displayed from database.m
- modelselect_Callback : changes the brand, dimension, price, type and color of the model variable according to the model the user selects; also displays them underneath the model dropdown list
- saveBOM_Callback : saves the bill of materials as a text file, with the file name and location of the user's choice

savePicture_Callback : saves the map image with the grid from user's input as a png file, with the file name and location of the user's choice
 saveText_Callback : saves the grid as a text file, with the file name and location of the user's choice
 loadText_Callback : loads the grid from the text file that the user previously saves
 reset_Callback : resets the grid so that no panels/turbines are placed on the map
 help_Callback : displays the message when the "help button" is clicked
 help_CreateFcn : replaces the default pushbutton figure with an icon of our choice
 findCity(handles, name) : finds the specified city index in class.m
 updateEstimate(handles): updates the energy estimate data using the functions in class.m

File **GUI.fig**

This contains the actual layout of the GUI. GUI.m uses this figure to display the actual information and allow user input, such as mouse clicks, dropdown menu selections, and save/load options.



File **icon.jpg**

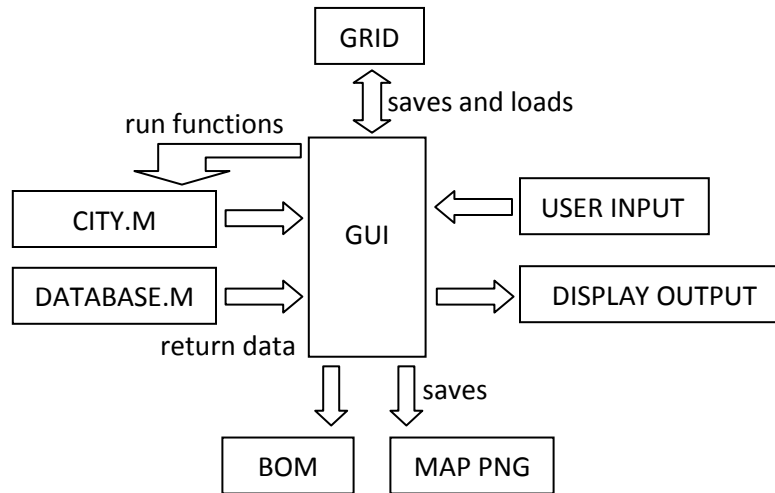
This is the icon used for the help button.



3. How the Program works

The GUI deals with the user input and data output, and other m-files are for support and data storing. While class.m and database.m contains the data for model, grid, and energy estimate, the GUI displays the data relevant to the specific city and model that the user is working with. It also directly saves the map png-file and text file, without going through other m-files.

Flow Chart:



4. Problems encountered and solutions

We encountered many problems during the development of the GUI.

The first problem that our group faced was figuring out how to overlay the images of the various cities on to the GUI when a city was chosen by the user. This issue was solved by loading the various pictures on the GUI's creation and displaying them when they were selected by the user.

Another problem that our group faced was the decision of using only three cities or the entire database of cities in the GUI. In the beginning, we wanted to include all of the cities. If we used all of the cities, the GUI would be a more realistic program for actual use. However, the file would exceed one hundred megabytes and suffer some lag since it would need to load all of the map data upon being opened. We decided to go with three cities for the sake of speed since that was the criteria of the project.

In the original design of the GUI the system was based on efficiency, so saving and loading files was based on left and right clicking which would not have been known if a person breezed over the instructions of the program. To deal with this the GUI was reformatted adding additional buttons and moving around some code to make the interface more user friendly.

5. Appendix

Break Down of the Tasks

Milestone description	Task(s)	Member(s)	Date
Class implementation	City.m/database.m Class coding	Wesley	3/8/13
GUI/IO design	GUI layout	Wesley, Nicholas	3/10/13
GUI/IO implementation	GUI coding, Google map images	Wesley, Jiun	3/13/13
Test and Debug	Test different cases. Fix bugs	Wesley, Jiun	3/15/13
Video	Video presentation	Jiun, Nicholas, Wesley	3/16/13

Brief Personal Summary

Wesley:

I wrote the class coding for the class implementation. I wrote the code for the database and city class including the constants, variables, methods, and constructor. For GUI/IO design, I did the GUI layout. I created and placed the objects onto the GUI. These objects included an axis object for the grid/map, a list box for model selection, a pop-up menu for city selection, static text to display energy data and model info, and buttons for the picture, BOM, and text output. For Testing and Debugging, I tested different cases as well for the GUI and also find and fix bugs. For the video presentation, I went over the source code of database.m and city.m, demonstrating their functionalities.

Jiun Rong:

To obtain the map images, I went to Google Map and took screen shots of specific cities for analyze. Then I wrote a function to create the images with a little gray scheme for clarity when it displayed with the grids. I compiled pricing.mat, database.mat, and retrieve weather.mat and allow database.m to be able to retrieve them. I added a few more functions in database.m and city.m for the GUI to run properly. The codes for each GUI function were written with the data and functions from the class m-file. After finishing the codes, I ran the GUI and create as many possible scenarios of what the user might input and make code changes as necessary. This resulted in the redesign of the interface to make it more user friendly. For the video presentation, I created a structure pictures detailing how the process of our program works, along with video recordings of the program run-throughs. I also did the editing of the video.

Nicholas:

For the GUI layout, I provided input and feedback on how the layout would work in efficiency and aesthetics. I explored other different arrangements for the GUI through experimentation to find a balance between efficiency, clarity, and comfort. For the video presentation, I took care of presenting the flow chart and project introduction areas of the film. I also acquired the equipment necessary to film for various sections of the video. For the report, I edited all of the various parts to make sure that the basic requirements were met and that it made sense and wrote the problems encountered section.

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

Wesley Ho, Jiun Rong Chen, Nicholas Yee-Toy

6. Sources

Image sources:

<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRZB9UtCbcIFSA18uWWEgsdJAdljK9RrMHdwKKku d2Y7pz-2stR>

<http://solartribune.com/wp-content/uploads/2011/07/etftrends.com-solar-radiation.jpg>