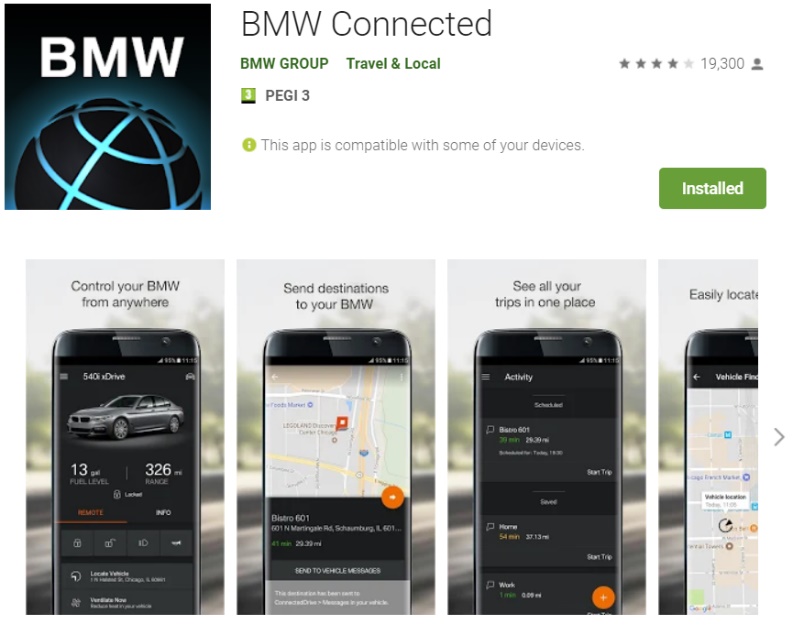
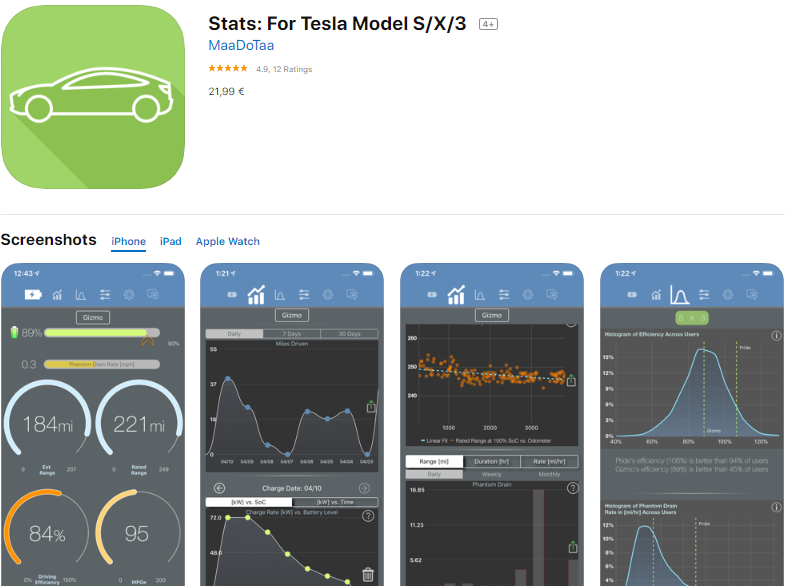
# Problem Definition

How far an electric car can go depending on a certain battery percent. Compares 'fuel' prices by taking average prices off a website. Calculates the impact of temperature and weather on range and journey efficiency. The software is targeted for car users but more specifically electric car users; there is very little software available that can provide this data.

# Research

I looked for apps to give information on cost savings, journey efficiency, effect of weather and climate on range and checking range. One of the apps I found “Stats: For Tesla Model S/X/3” where it performs similar functions to my intended software but is only for tesla and won’t take in to account any weather variability. Another app, “BMW Connected” had poor reviews and very few features. I searched for the best website to get average UK fuel prices; I based the ‘quality’ of the website on how easy it would be to take the values off it. The website I chose was <https://www.allstarcard.co.uk/tools/uk-fuel-prices/> as the average fuel price values are updated regularly so would add the accuracy of my software as well as being easy to acquire the values. I also spoke to electric vehicle users and asked for their feedback on the app built for their car with the main downside being a lack of features available and its unreliability. They also inputted what features they thought may be useful as well as trying to make the different vehicle types colour coded so it could be differentiated from the menu options. I read some articles talking about the effect HVAC has on an electric battery and the range it takes off the journey and used those mulitpliers in my weather objects.

# Solution Description

My software will allow users to calculate journey efficiency by comparing your actual range to the range you achieved during a particular journey using the amount of battery percent / fuel used, your range depending on battery percent by doing range / 100 \* battery % (all results will be in miles) while considering weather by allowing the user to select which weather type they want active and assuming the effect on range if you have wipers on or HVAC for example. It also calculates price per mile of different vehicles by taking average fuel prices off a website (https://www.allstarcard.co.uk/tools/uk-fuel-prices/) and calculating your price per mile as well as being able to create your own vehicle to simulate if yours is not there which will also be saved on the text file. You can select the vehicle you want to simulate as well as default weather which is saved in a text file, so they don’t have to constantly be reset. Data on the available vehicles are stored on a text file, range, electric range, engine range, fuel type, name, tank size and battery size which will be loaded in each time the software is run. This data can also be sorted in order of range in settings so when new vehicles are added you can see which has the highest range. You will also be able to print all vehicle and weather statistics so the user can see all the properties. The user will also be able to reset the vehicle data and settings text files as well as the ability to update the restore text file.

public void LoadVehicleData()

This subroutine puts the Vehicle data.txt contents and puts it into an array. From this data it gets the amount of each vehicle so the for loops know how long to run for. Then all the information for the text file is set to the vehicles using the corresponding classes 'setters'.

public void LoadSettings()

This loads all the settings data from the Settings.txt file to be used in the software so the user wouldn’t have to change their default weather and vehicle each time the program was loaded.

public void SetPrices()

This subroutine downloads a HTML file from a website (“https://www.allstarcard.co.uk/tools/uk-fuel-prices/”) which has information about the average fuel prices in the UK. I find where the values are within the HTML code and then can pinpoint it using if statements to check whether it's in-between certain tags and then 'finds' the values. I then set those values into an array where I remove all other non-numeric characters using the regex expression "^0-9." and then using the replace syntax. This is because the raw HTML file has lots of excessive characters which can corrupt the values. Using my fuel prices class, I set the values to be used in other calculations available for the user such as working on price per mile.

public void Start()

Calls the subroutine DisplayData() and then allows the user to select an option from the menu.

public void DispalyData()

Displays the vehicle data list (just the vehicle names) as well as the current default weather and current default vehicle by getting that information off the Settings.txt file.

public void Settings()

Allows the user to change the default vehicle, default weather, to create a vehicle to simulate, to view vehicle statistics and to restore all settings and vehicles available to default. Default vehicle and weather use the corresponding classes to set the values. It then writes all these values to the settings.txt file

public void SaveSettings()

This subroutine is sent the settings data from the Settings subroutine and then saves them to a text file to be used in future sessions to save the hassle of having to reset them each time.

public void Statistics()

Displays all vehicle statistics and range deductions for each weather so the user can check all the properties so they can pick a vehicle with similar statistics to their vehicle if they don’t want to create their own vehicle to simulate.

public double RangeCheck()

Allows the user to enter their current EV battery percent and then calculates its expected distance to travel by taking the selected vehicle range (default unless not an electric vehicle then will ask you to select one), divides it by 100, multiplies by entered battery percent and then multiplies by the weather range deduction value.

public string PriceCompare()

Compares the price per mile of a selected electric vehicle vs engine vehicle by doing battery size / range \* average electric price for electric vehicles and for engine vehicles, it gets the fuel type of the selected vehicle and then takes tank size / range \* fuel price. Cannot perform calculations for hybrid vehicles as it will be difficult to accurately merge the different efficiency of battery and the engine.

public string JourneyEfficiency()

Allows the user to see how efficient their journey was by calculating expected distance to travel and comparing it to the user’s distance travelled. It asks the user to enter how far they travelled and how much fuel / battery % they used (will use default vehicle) and depending how far off they were from the expected journey efficiency they were it will display a different message so for example if the actual journey efficiency was way below expected, it will return "Your journey was very inefficient". Cannot perform calculations for hybrid vehicles as it will be difficult to accurately merge the different efficiency of battery and the engine.

public void CreateVehicle()

This subroutine allows the user to create vehicles with custom properties that you can use to simulate, it then rewrites the text file that stores all the vehicle properties to add the new vehicle to be used in future simulations. First the user will enter their vehicle type, eg engine and then it shifts all the values forward in the vehicleData array by however many new values that will need to be added. Then it runs through asking you to enter all the properties of that vehicle and storing them in the vehicleData array in the correct position corresponding to the vehicle type. It then writes the appended array to the Vehicle data.txt file and then calls the LoadVehicleData subroutine to reload all the data.

public void RestoreToDefault()

This lets the user restore all the vehicle and settings data to default by having a restore file. It will delete the original data files and replace them with the restore files and then reloads settings and vehicle data which can be used if the user wanted to add some temporary vehicles to simulate but didn’t want them in the software permanently.

public void UpdateRestore()

This deletes the restore file and replaces it with the current Vehicle data.txt file so that if the user would want to restore to defaults then they can keep cars they created previously which is useful if the user just wanted to create some test vehicles for permanent use.

public void SortVehicleData(int enCarAmount, int elCarAmount, int hyCarAmount)

Uses a bubble sort to sort the vehicle data from lowest range to highest within the vehicle types. As it must move multiple values, it uses a for loop within the bubble sort to shift the rest of the values. It also has to reset the index as after sorting vehicle data the indexes change so need to be able to update default vehicle.

public void IntValidation(string data, string exceptionErrorMSG, int min, int max)

Validates string to integer conversions as I must convert all the inputs constantly, I made a sub routine, so I don't have to do it several times separately. It takes minimum and maximum values as well so it can validate input ranges too such as battery percent where the value can’t be over 100. It will then print the error messages in red to stand out and the subroutine will only exit if the value entered is correct.

For all calculations (except PriceCompare), it will always use the vehicle and weather that’s active which can be changed in settings. By default, the weather is set to default and vehicle is “BMW i3 94Ah” which is the set values in the “Settings restore.txt” file.

If you create a vehicle with thee name {5} or any number above 0 inside the brackets, it breaks.

# Requirements

1 – There will be 3 different car types, “engine”, “electric” and “hybrid” which will be objects

* 1. – Each car will be inherited from the superclass “car”
  2. – All cars will have a range and name value, the electric cars will store battery size, the engine cars will store road tax, fuel type and tank capacity and the hybrid cars will store all the variables from each car as well as electric and engine range

1. – There will 5 different weather types
   1. – Each weather will store a range deduction multiplier, a name and a “isOn” value to set which weather is active
   2. – Each weather will change the distance able to be travelled by creating rough deductions to the available range by considering HVAC
2. – Almost all the data and settings will be read off a text file
   1. – All the vehicle data is stored, where the top of the text file states the amount of each vehicle that is going to be loaded, then each vehicle and its corresponding values in their vehicle types
   2. – The previous settings are stored, where the default selected weather index is stored followed by the default selected vehicle type and then index
3. – The fuel prices will be taken off <https://www.allstarcard.co.uk/tools/uk-fuel-prices/>
   1. – The webpage will be downloaded in pure HTML form and then stripped to acquire the necessary values
4. – The settings option will display a list of options for the user
   1. – The user will be able to change the default weather used by the software and it will then refresh the menu
   2. - The user will be able to change the default vehicle used by the software and it will then refresh the menu
   3. – The user will be able to view all the vehicle statistics and weather information with appropriate units as well as being colour coded
   4. – The user will be able to sort all vehicles within their groups in order of range (highest to lowest) using a bubble sort
   5. – The user will be able to create their own vehicle to simulate by entering in the required properties for each vehicle and then will be written to the vehicle data text file
5. – There will be a function to calculate range depending on battery % and or fuel
   1. – The software will use the default vehicle and then take that vehicles range / 100 \* battery percent and or (fuel amount / tank size) \* weather multiplier
6. – There will be a function to compare price per mile of 2 different vehicles
   1. - The user will choose 2 vehicles and the software and using the electric price per mile and fuel price per mile of the vehicle by using its range and battery capacity and or tank size and then using the predefined electric price per kWh and diesel and petrol price per litre
7. – There will be a function to determine journey efficiency
   1. – using the software’s

# Structure Diagram

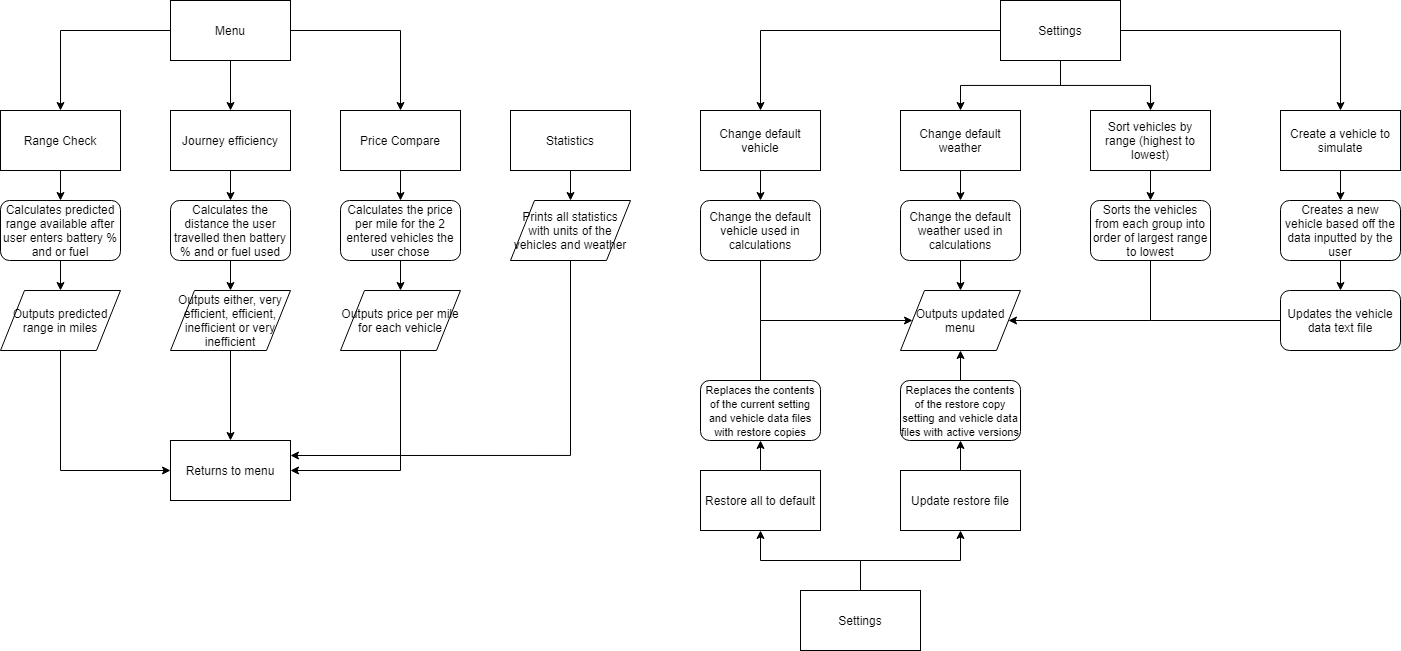
|  |
| --- |
| Cars |
| # string name |
| # double range |

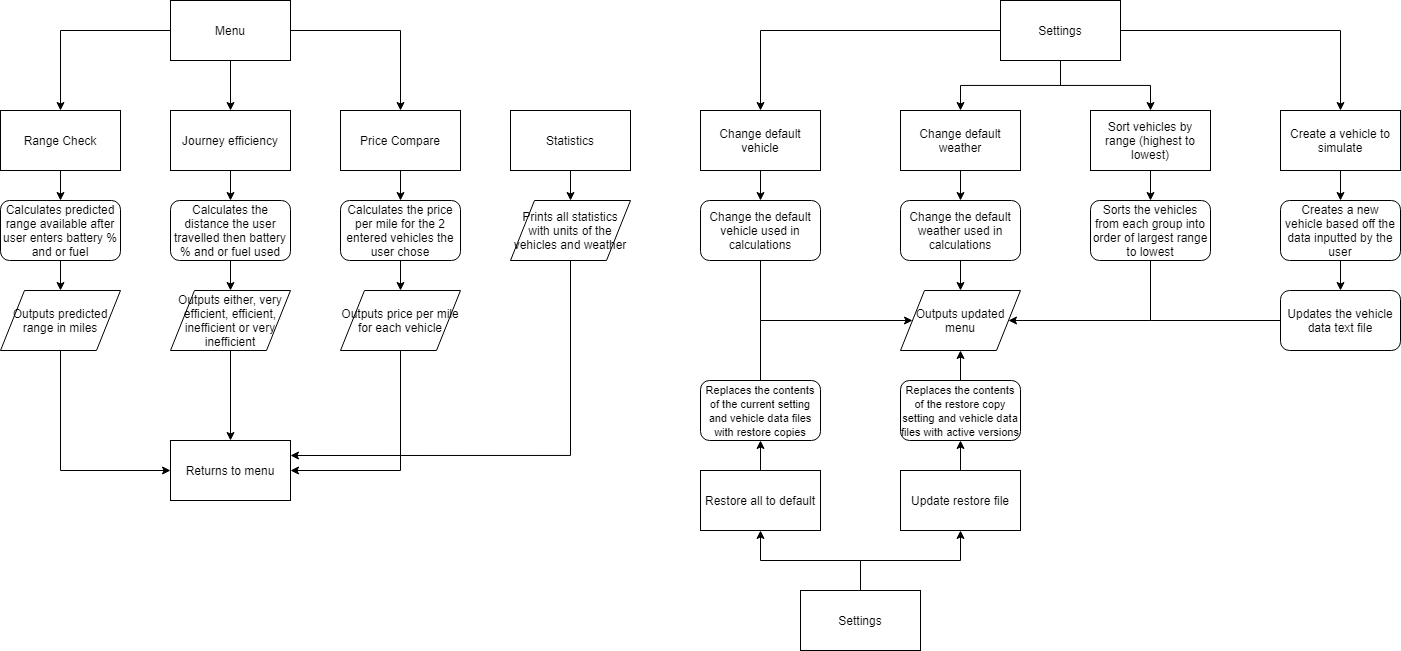
|  |
| --- |
| HybridCars: Cars |
| - double tankSize |
| - double batterySize |
| - double enigneRange |
| - char fuelType |

|  |
| --- |
| ElectricCars: Cars |
| - double batterySize |

|  |
| --- |
| EngineCars: Cars |
| - double roadTax |
| - double tankSize |
| - char fuelType |

# Overall Flowchart





# User Interface Design

# Description of Data Structures

There are 3 different arrays to store vehicle data for the 3 different types, each object has an array where properties are inherited from a superclass to store the name and range of each vehicle as they are common to all the car sub classes, then each subclass has their unique variables such as tank size, battery size or fuel type. All the data for the arrays is filled in from the vehicle data text file. The array sizes are 1 length larger than the number of vehicles listed so when a new vehicle is added, it won’t exceed boundaries of the array.

The different weather types (5) are stored in an array of the weather object which stores the name, whether it’s active and the range reduction multiplier.

The fuel prices variable stores the average petrol, diesel price and average electric price per kWH

The vehicle data text file stores the amount of each vehicle type (how many vehicles it will be loading) followed by the corresponding vehicle data starting with the vehicle name, in the order engine vehicles, electric vehicles and then hybrid vehicles.

The settings data text file stores the index of the weather that was being used to be used in the array, the default vehicle type and the default vehicle index.

Both have restore copies.

# Commented Program Code

**class Program**

static void Main(string[] args)

{

Console.Title = ("Vehicle statistics and calculations");

Run run = new Run();

}

**class Run**

private EngineCars[] EnCar;

private HybridCars[] HyCar;

private ElectricCars[] ElCar;

private Settings SettingsData;

private FuelPrices FP;

private Weather[] Weathers = new Weather[5];

string[] vehicleData = new string[@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt".Length + 10];

**Takes the vehicle data from the VehicleData text file and sets the object properties**

public void LoadVehicleData()

{

int k = 3;

string[] uploadData = File.ReadAllLines(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt");

for (int i = 0; i < uploadData.Length; i++)

{

vehicleData[i] = uploadData[i];

}

int engineCarAmount = Convert.ToInt32(vehicleData[0]); //so it knows how many times it has to loop while loading data of the text file, the start of the file contains the amount of each vehicle

int electricCarAmount = Convert.ToInt32(vehicleData[1]);

int hybridCarAmount = Convert.ToInt32(vehicleData[2]);

EnCar = new EngineCars[engineCarAmount + 1]; //has to be instantiated here because I need the sizes to change as you can add more cars so the +1 is preparing oncase that happens

ElCar = new ElectricCars[electricCarAmount + 1];

HyCar = new HybridCars[hybridCarAmount + 1];

for(int i = 0; i < engineCarAmount; i++) //instantiates all vehicles

{

EnCar[i] = new EngineCars("", 0, 0, 0, ' ');

}

for (int i = 0; i < electricCarAmount; i++)

{

ElCar[i] = new ElectricCars("", 0, 0);

}

for (int i = 0; i < hybridCarAmount; i++)

{

HyCar[i] = new HybridCars("", 0, 0, 0, ' ', 0, 0);

}

for (int index = 0; index < engineCarAmount; index++) //gets all the vehicle data and sets that for the vehcile objects

{

EnCar[index].SetName(vehicleData[k]);

k++;

EnCar[index].SetRange(Convert.ToInt32(vehicleData[k]));

k++;

EnCar[index].SetTankSize(Convert.ToDouble(vehicleData[k]));

k++;

EnCar[index].SetRoadTax(Convert.ToInt32(vehicleData[k]));

k++;

EnCar[index].SetFuelType(Convert.ToChar(vehicleData[k]));

k++;

}

for (int index = 0; index < electricCarAmount; index++)

{

ElCar[index].SetName(vehicleData[k]);

k++;

ElCar[index].SetRange(Convert.ToInt32(vehicleData[k]));

k++;

ElCar[index].SetBatterySize(Convert.ToDouble(vehicleData[k]));

k++;

}

for (int index = 0; index < hybridCarAmount; index++)

{

HyCar[index].SetName(vehicleData[k]);

k++;

HyCar[index].SetRange(Convert.ToInt32(vehicleData[k]));

k++;

HyCar[index].SetTankSize(Convert.ToDouble(vehicleData[k]));

k++;

HyCar[index].SetBatterySize(Convert.ToDouble(vehicleData[k]));

k++;

HyCar[index].SetFuelType(Convert.ToChar(vehicleData[k]));

k++;

HyCar[index].SetElectricRange(Convert.ToDouble(vehicleData[k]));

k++;

HyCar[index].SetEngineRange(Convert.ToDouble(vehicleData[k]));

k++;

}

}

**Loads all the default settings information off the text file so you don't have to keep resetting them**

public void LoadSettings()

{

string[] data = File.ReadAllLines(@"E:\College Y2\Computer Science\Coursework\Settings.txt");

SettingsData.SetWeatherIndex(Convert.ToInt32(data[0]));

SettingsData.SetVehicleType(data[1]);

SettingsData.SetVehicleIndex(Convert.ToInt32(data[2]));

}

**Takes the average fuel prices off the website below**

public void SetPrices()

{

WebClient client = new WebClient();

string petrolPrice;

string dieselPrice;

try

{

string downloadString = client.DownloadString("https://www.allstarcard.co.uk/tools/uk-fuel-prices/");

string[] html = downloadString.Split('\n');

string[] prices = new string[4];

int j = 0;

for (int i = 0; i < html.Length; i++) //finds the values because it's a certain amount of lines between the <h2> tags

{

if (html[i].Contains("<h2>"))

{

if (html[i + 2].Contains("</h2>"))

{

prices[j] = html[i + 1];

j++;

}

}

}

petrolPrice = prices[0];

dieselPrice = prices[1];

Regex rgx = new Regex("[^0-9.]");

petrolPrice = rgx.Replace(petrolPrice, "");

dieselPrice = rgx.Replace(dieselPrice, ""); //strips the whole raw html text down so it's just the values and no extra spaces etc

}

catch(Exception)

{

DialogResult input = MessageBox.Show("Unable to reach allstarcard.co.uk/tools/uk-fuel-prices/ to retrieve fuel price information, would you like to enter your own fuel prices? (if not, rough values will be used instead)", "Error", MessageBoxButtons.YesNo);

if(DialogResult.Yes == input)

{

Console.Write("Enter petrol price in P per Litre: ");

petrolPrice = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a suitable value", 1, 1000));

Console.Write("Enter diesel price in P per Litre: ");

dieselPrice = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a suitable value", 1, 1000));

Console.Clear();

}

else

{

petrolPrice = "125.5";

dieselPrice = "129.1";

}

}

FP.SetPetrolPrice(Convert.ToDouble(petrolPrice));

FP.SetDieselPrice(Convert.ToDouble(dieselPrice));

}

**Displays the option menu**

public void Start()

{

int input = 0;

while (input != 5)

{

DisplayData();

Console.WriteLine("Please select one of the following options (1-5):");

Console.WriteLine("1 - Range check");

Console.WriteLine("2 - Running costs of an electric vehicle compared to a engine per mile");

Console.WriteLine("3 - Journey efficiency (only for engine and electric vehicles)");

Console.WriteLine("4 - Settings");

Console.WriteLine("5 - Exit");

input = IntValidation(Console.ReadLine(), "Please enter a number between 1 and 5", 1, 5);

switch (input)

{

case 1:

Console.WriteLine("You can travel roughly " + RangeCheck() + " miles");

break;

case 2:

Console.WriteLine(PriceCompare());

break;

case 3:

Console.WriteLine(JourneyEfficiency());

break;

case 4:

Settings();

break;

case 5:

Console.WriteLine("Exiting Program");

break;

}

Console.WriteLine("\n" + "Press enter to continue");

Console.ReadLine();

Console.Clear();

}

}

**Displays the data at the start of the program including the default settings**

public void DisplayData()

{

int engineCarAmount = Convert.ToInt32(vehicleData[0]);

int electricCarAmount = Convert.ToInt32(vehicleData[1]);

int hybridCarAmount = Convert.ToInt32(vehicleData[2]);

Console.WriteLine("Use this software to work out expenses of running an electric car vs an engine car");

Console.Write("Current default vehicle is: ");

Console.ForegroundColor = ConsoleColor.DarkCyan;

if (SettingsData.GetVehicleType() == "ENGINE") //gets the default vehicle name

{

Console.WriteLine(EnCar[SettingsData.GetVehicleIndex()].GetName());

}

if (SettingsData.GetVehicleType() == "ELECTRIC")

{

Console.WriteLine(ElCar[SettingsData.GetVehicleIndex()].GetName());

}

if (SettingsData.GetVehicleType() == "HYBRID")

{

Console.WriteLine(HyCar[SettingsData.GetVehicleIndex()].GetName());

}

Console.ResetColor();

Console.Write("\nWeather is set to: ");

Console.ForegroundColor = ConsoleColor.DarkCyan; //changes colour so it can be easier differentiated from the menu options

Console.WriteLine(Weathers[SettingsData.GetWeatherIndex()].GetName() + "\n"); //writes the default weather

Console.ResetColor();

Console.WriteLine("Current cars available:");

Console.ForegroundColor = ConsoleColor.DarkMagenta;

Console.WriteLine("Engine:");

for(int i = 0; i < engineCarAmount; i++)

{

Console.WriteLine(i + " - " + EnCar[i].GetName());

}

Console.Write("\n\n");

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("Electric:");

for (int i = 0; i < electricCarAmount; i++)

{

Console.WriteLine(i + " - " + ElCar[i].GetName());

}

Console.Write("\n\n");

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("Hybrid:");

for (int i = 0; i < hybridCarAmount; i++)

{

Console.WriteLine(i + " - " + HyCar[i].GetName());

}

Console.Write("\n\n");

Console.ResetColor();

}

**Controls the menu for the settings**

public void Settings()

{

Console.WriteLine("\n");

int selectedWeatherIndex = -1;

int selectedVehicleIndex = -1;

string vehicleType = "";

Console.WriteLine("Settings - Please select one of the following options:");

Console.WriteLine("1 - Change default weather");

Console.WriteLine("2 - Change default vehicle");

Console.WriteLine("3 - Statistics");

Console.WriteLine("4 - Sort vehicles by range (highest to lowest)");

Console.WriteLine("5 - Create a vehicle to Simulate");

Console.WriteLine("6 - Restore all to default");

Console.WriteLine("7 - Update restore file");

Console.WriteLine("8 - Exit");

int input = IntValidation(Console.ReadLine(), "Select an option between 1 and 8", 1, 8);

switch (input)

{

case 1: //allows the user to change the default weather

Console.WriteLine("These are the current supported weathers available, select either a number 1-5 to choose which weather you want active:");

Console.WriteLine("1 - Default");

Console.WriteLine("2 - Rain");

Console.WriteLine("3 - Snow");

Console.WriteLine("4 - Ice");

Console.WriteLine("5 - Sun");

Console.WriteLine("6 - Exit");

selectedWeatherIndex = IntValidation(Console.ReadLine(), "Please select which weather you want to select, 1 - 5", 1, 5) - 1;

Weathers[SettingsData.GetWeatherIndex()].SetOn(false);

Weathers[selectedWeatherIndex].SetOn(true);

Console.WriteLine("You have selected " + Weathers[selectedWeatherIndex].GetName());

break;

case 2: //allows the user to set a default vehicle

Console.WriteLine("Enter the desired vehicle type (eg Electric) followed by the vehicle number from the list above:");

vehicleType = Console.ReadLine().ToUpper();

while (vehicleType != "ENGINE" && vehicleType != "ELECTRIC" && vehicleType != "HYBRID")

{

Console.WriteLine("Enter one of the listed vehicle types: \n Engine \n Electric \n Hybrid");

vehicleType = Console.ReadLine().ToUpper();

}

if (vehicleType == "ENGINE")

{

selectedVehicleIndex = IntValidation(Console.ReadLine(), "Please select an engine vehicle, 0 - " + EnCar.Length + " from the listed above", 0, EnCar.Length - 1);

Console.WriteLine("You have selected " + EnCar[selectedVehicleIndex].GetName(), " as the default vehicle");

}

if (vehicleType == "ELECTRIC")

{

selectedVehicleIndex = IntValidation(Console.ReadLine(), "Please select an electric vehicle, 0 - " + ElCar.Length + "from the listed above", 0, ElCar.Length - 1);

Console.WriteLine("You have selected " + ElCar[selectedVehicleIndex].GetName() + " as the default vehicle");

}

if (vehicleType == "HYBRID")

{

selectedVehicleIndex = IntValidation(Console.ReadLine(), "Please select a hybrid vehicle, 0 - " + HyCar.Length + " from the listed above", 0, HyCar.Length - 1);

Console.WriteLine("You have selected " + HyCar[selectedVehicleIndex].GetName() + " as the default vehicle");

}

break;

case 3: //allows the user to view all vehicle and weather values such as range reducer from weather

Statistics();

break;

case 4: //sorts all vehicles from lowest range to highest within their vehicle types

SortVehicleData(Convert.ToInt32(vehicleData[0]), Convert.ToInt32(vehicleData[1]), Convert.ToInt32(vehicleData[2]));

break;

case 5: //allows the user to create their own custom vehicle

CreateVehicle();

break;

case 6:

RestoreToDefault();

break;

case 7:

UpdateRestore();

break;

case 8:

Console.WriteLine("Exiting settings");

break;

}

SaveSettings(selectedWeatherIndex, vehicleType, selectedVehicleIndex); //saves anything that may have been changed

}

**Saves the current settings data to the Settings text file**

public void SaveSettings(int weatherIndex, string vehicleType, int defaultVehicle) //updates the settings text file

{

if (weatherIndex == -1)

{

weatherIndex = SettingsData.GetWeatherIndex();

}

if (defaultVehicle == -1)

{

defaultVehicle = SettingsData.GetVehicleIndex();

vehicleType = SettingsData.GetVehicleType();

}

string[] saveData = {Convert.ToString(weatherIndex), vehicleType, Convert.ToString(defaultVehicle)}; //puts the settings data in an array to be written to a text file

File.WriteAllLines(@"E:\College Y2\Computer Science\Coursework\Settings.txt", saveData);

LoadSettings();

}

**Prints out all the vehicle data while adding headers to make it understandable to the user**

public void Statistics()

{

Console.WriteLine("Here are the following statistics about the weather and vehicles");

Console.ForegroundColor = ConsoleColor.DarkCyan;

Console.WriteLine("Range deductions for each weather (These represent the percent it will take off based on HVAC and effect on batteries)");

for(int i = 0; i < Weathers.Length; i++)

{

Console.WriteLine(Weathers[i].GetName() + " - " + Weathers[i].GetRangeDeduction());

}

Console.ForegroundColor = ConsoleColor.DarkMagenta;

Console.WriteLine("\nEngine vehicle data:");

for(int i = 0; i < EnCar.Length - 1; i++)

{

Console.WriteLine("Name " + EnCar[i].GetName());

Console.WriteLine("Range (miles) " + EnCar[i].GetRange());

Console.WriteLine("Tank size (litres) " + EnCar[i].GetTankSize());

Console.WriteLine("Road tax (GBP) " + EnCar[i].GetRoadTax());

Console.WriteLine("Fuel type (P for petrol, D for diesel) " + EnCar[i].GetFuelType() + "\n");

}

Console.ForegroundColor = ConsoleColor.DarkYellow;

Console.WriteLine("\nElectric vehicle data:");

for (int i = 0; i < ElCar.Length - 1; i++)

{

Console.WriteLine("Name " + ElCar[i].GetName());

Console.WriteLine("Range (miles) " + ElCar[i].GetRange());

Console.WriteLine("Battery Size (kWH) " + ElCar[i].GetBatterySize() + "\n");

}

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("\nHybrid vehicle data:");

for (int i = 0; i < HyCar.Length - 1; i++)

{

Console.WriteLine("Name " + HyCar[i].GetName());

Console.WriteLine("Range (miles) " + HyCar[i].GetRange());

Console.WriteLine("Tank size (L) " + HyCar[i].GetTankSize());

Console.WriteLine("Battery size (kWH) " + HyCar[i].GetBatterySize());

Console.WriteLine("Fuel type (P for petrol, D for diesel) " + HyCar[i].GetFuelType());

Console.WriteLine("Electric range (Miles) " + HyCar[i].GetElectricRange());

Console.WriteLine("Engine range (Miles) " + HyCar[i].GetEngineRange() + "\n");

}

Console.ResetColor();

}

**Calculates your rough expected range depending on battery percent and weather**

public double RangeCheck()

{

double range = 0;

if(SettingsData.GetVehicleType() == "ELECTRIC")

{

range = ElCar[SettingsData.GetVehicleIndex()].GetRange(); //gets vehicle range

Console.WriteLine("Enter your current battery percent to the nearest whole number");

double batteryPercent = IntValidation(Console.ReadLine(), "Please enter a battery percent, 1 - 100%", 1, 100);

range = range / 100 \* batteryPercent \* Weathers[SettingsData.GetWeatherIndex()].GetRangeDeduction(); // calculates range by doing actual range / 100 \* battery percent \* range deduction from active weather

}

if(SettingsData.GetVehicleType() == "ENGINE")

{

range = EnCar[SettingsData.GetVehicleIndex()].GetRange();

Console.WriteLine("Enter the rough amount of fuel in your vehicle in litres");

double fuelAmount = IntValidation(Console.ReadLine(), "Enter a suitable fuel amount for your vehicle", 1, Convert.ToInt32(EnCar[SettingsData.GetVehicleIndex()].GetTankSize()));

range = range / 100 \* (fuelAmount / EnCar[SettingsData.GetVehicleIndex()].GetTankSize() \* 100) \* Weathers[SettingsData.GetWeatherIndex()].GetRangeDeduction(); //calculates range by doing vehicles actual range / tank size \* 100 \* range deduction from active weather

}

if(SettingsData.GetVehicleType() == "HYBRID")

{

double enRange = HyCar[SettingsData.GetVehicleIndex()].GetEngineRange();

double elRange = HyCar[SettingsData.GetVehicleIndex()].GetElectricRange();

Console.WriteLine("Enter the rough amount of fuel in your vehicle in litres");

double fuelAmount = IntValidation(Console.ReadLine(), "Enter a suitable fuel amount for your vehicle", 1, Convert.ToInt32(HyCar[SettingsData.GetVehicleIndex()].GetTankSize()));

Console.WriteLine("Enter your current battery percent to the nearest whole number");

double batteryPercent = IntValidation(Console.ReadLine(), "Please enter a battery percent, 1 - 100%", 1, 100);

range = Math.Round((enRange / 100 \* (fuelAmount / HyCar[SettingsData.GetVehicleIndex()].GetTankSize() \* 100) \* Weathers[SettingsData.GetWeatherIndex()].GetRangeDeduction()) + (elRange / 100 \* batteryPercent \* Weathers[SettingsData.GetWeatherIndex()].GetRangeDeduction()));

}

return range;

}

**compares price per mile for an electric and an engine car by using the average fuel price per litre**

public string PriceCompare()

{

int selectedElectricCar;

int selectedEngineCar;

double costPerMileFuel;

Console.WriteLine("Select an electric vehicle by entering the index (number to the left) of the vehicle from the list above");

selectedElectricCar = IntValidation(Console.ReadLine(), "Please select an electric vehicle, 0 - " + Convert.ToString(Convert.ToInt32(vehicleData[1])-1), 0, Convert.ToInt32(vehicleData[0]) - 1);

Console.WriteLine("Select an engine vehicle by entering the index (number to the left) of the vehicle from the list above");

selectedEngineCar = IntValidation(Console.ReadLine(), "Please select an engine vehicle, 0 - " + Convert.ToString(Convert.ToInt32(vehicleData[0])-1), 0, Convert.ToInt32(vehicleData[0]) - 1) ;

double costPerMileElectric = ElCar[selectedElectricCar].GetBatterySize() / ElCar[selectedElectricCar].GetRange() \* FP.GetElectricPrice(); //get price per mile from battery size and range

if (EnCar[selectedEngineCar].GetFuelType() == 'D') //calculates cost per mile for the engine vehicle depending on fuel type buy doing the vehicles tank size / range to get miles per litre then multiplying by fuel price

{

costPerMileFuel = EnCar[selectedEngineCar].GetTankSize() / EnCar[selectedEngineCar].GetRange() \* FP.GetDieselPrice();

}

else

{

costPerMileFuel = EnCar[selectedEngineCar].GetTankSize() / EnCar[selectedEngineCar].GetRange() \* FP.GetPetrolPrice();

}

costPerMileElectric = Math.Round(costPerMileElectric, 2);

costPerMileFuel = Math.Round(costPerMileFuel, 2);

string send = "It will cost you roughly " + costPerMileFuel + " pence per mile using " + EnCar[selectedEngineCar].GetName() + "\nwhereas it costs roughly "

+ costPerMileElectric + " pence per mile using " + ElCar[selectedElectricCar].GetName();

return send;

}

**Calculates journey efficiency by working out expected battery / fuel usage for the journey and comparing it to the actual efficiency the driver got**

public string JourneyEfficiency()

{

int distanceTravelled;

int batteryUsed;

int fuelUsed;

double perfectEfficiency;

double actualEfficiency;

Console.WriteLine("How far did you travel in miles?");

distanceTravelled = IntValidation(Console.ReadLine(), "Please enter how far you travelled in miles", 1, 800);

switch (SettingsData.GetVehicleType())

{

case "ENGINE":

Console.WriteLine("Roughly how much fuel did you use in litres");

fuelUsed = IntValidation(Console.ReadLine(), "Please enter a suitable value for your fuel consumption", 1, 128);

perfectEfficiency = EnCar[SettingsData.GetVehicleIndex()].GetRange() / EnCar[SettingsData.GetVehicleIndex()].GetTankSize();

actualEfficiency = distanceTravelled / fuelUsed;

if (actualEfficiency \* 0.9 > perfectEfficiency)

{

return "Your journey was very efficient";

}

if (actualEfficiency > perfectEfficiency)

{

return "Your journey was efficienct";

}

if (actualEfficiency < perfectEfficiency)

{

return "Your journey was inefficient";

}

if (actualEfficiency < perfectEfficiency \* 0.9)

{

return "Your journey was very inefficient";

}

return null;

case "ELECTRIC":

Console.WriteLine("Roughly what battery percent did you use?");

batteryUsed = IntValidation(Console.ReadLine(), "Please enter the battery percent you used for your journey", 1, 100);

perfectEfficiency = ElCar[SettingsData.GetVehicleIndex()].GetRange() / ElCar[SettingsData.GetVehicleIndex()].GetBatterySize(); //range per kWH

actualEfficiency = distanceTravelled / (ElCar[SettingsData.GetVehicleIndex()].GetBatterySize() / batteryUsed \* 100);

if (actualEfficiency > perfectEfficiency)

{

return "Your journey was very efficient";

}

if (actualEfficiency >= perfectEfficiency)

{

return "Your journey was efficienct";

}

if (actualEfficiency < perfectEfficiency)

{

return "Your journey was inefficient";

}

else

{

return "Your journey was very inefficient";

}

case "HYBRID":

Console.WriteLine("Roughly how much fuel did you use in litres");

fuelUsed = IntValidation(Console.ReadLine(), "Please enter a suitable value for your fuel consumption", 1, 128);

Console.WriteLine("Roughly what battery percent did you use?");

batteryUsed = IntValidation(Console.ReadLine(), "Please enter the battery percent you used for your journey", 1, 100);

perfectEfficiency = HyCar[SettingsData.GetVehicleIndex()].GetRange() / (HyCar[SettingsData.GetVehicleIndex()].GetTankSize() + HyCar[SettingsData.GetVehicleIndex()].GetBatterySize()); //range per kWH per L

actualEfficiency = distanceTravelled / (fuelUsed + (ElCar[SettingsData.GetVehicleIndex()].GetBatterySize() / batteryUsed \* 100));

if (actualEfficiency > perfectEfficiency)

{

return "Your journey was very efficient";

}

if (actualEfficiency >= perfectEfficiency)

{

return "Your journey was efficienct";

}

if (actualEfficiency < perfectEfficiency)

{

return "Your journey was inefficient";

}

else

{

return "Your journey was very inefficient";

}

default:

return null;

}

}

**Allows the user to create a vehicle to simulate**

public void CreateVehicle()

{

int engineCarAmount = Convert.ToInt32(vehicleData[0]);

int electricCarAmount = Convert.ToInt32(vehicleData[1]);

int hybridCarAmount = Convert.ToInt32(vehicleData[2]);

int engineDataLength = 5;

int electricDataLength = 3;

int hybridDataLength = 7;

string[] uploadData = File.ReadAllLines(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt");

int dataLength = (engineCarAmount \* engineDataLength) + (electricCarAmount \* electricDataLength) + (hybridCarAmount \* hybridDataLength);

for (int i = 0; i < uploadData.Length; i++)

{

vehicleData[i] = uploadData[i];

}

int index;

Console.WriteLine("Type a vehicle type: Engine, Electric or Hybrid");

string vehicleType = Console.ReadLine();

if (vehicleType.ToUpper().Contains("EN")) //collects all the vehicle data for that vehicle type except range and name as they are common of all vehicle types so are done seperately

{

index = 3; //after all the vehicle amount values

for (int i = dataLength + 3; i > 0; i--) //shifts everything forward in the array

{

vehicleData[i + engineDataLength] = vehicleData[i];

}

vehicleData[0] = Convert.ToString(Convert.ToInt32(vehicleData[0]) + 1);

Console.WriteLine("Type a name for your vehicle");

vehicleData[index] = Console.ReadLine();

Console.WriteLine("Enter the range of your vehicle");

vehicleData[index + 1] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic value for your vehicles range", 1, 1000));

Console.WriteLine("Enter the tank size (L), road tax and fuel type (either P or D) for your vehicle one after another");

vehicleData[index + 2] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic tank size for your vehicle", 1, 200));

vehicleData[index + 3] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a real road tax value for your vehicle", 0, 2135));

vehicleData[index + 4] = Console.ReadLine().ToUpper();

while (vehicleData[index + 4] != "P" && vehicleData[index + 4] != "D")

{

Console.WriteLine("Enter a suitable fuel type, P for petrol and D for diesel");

vehicleData[index + 4] = Console.ReadLine();

}

}

if (vehicleType.ToUpper().Contains("HY"))

{

index = 3 + (electricDataLength \* electricCarAmount) + (engineDataLength \* engineCarAmount); //so it knows where in the array to place it

for (int i = dataLength + 3; i > index - hybridDataLength; i--) //shifts everything forward in the array

{

vehicleData[i + hybridDataLength] = vehicleData[i];

}

vehicleData[2] = Convert.ToString(Convert.ToInt32(vehicleData[2]) + 1);

Console.WriteLine("Type a name for your vehicle");

vehicleData[index] = Console.ReadLine();

Console.WriteLine("Enter the range of your vehicle");

vehicleData[index+1] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic value for your vehicles range", 1, 1000));

Console.WriteLine("Enter the tank size in litres (L), battery size in kWh, fuel type (either P or D), the electric range and engine range in miles");

vehicleData[index+2] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic tank size for your vehicle", 1, 200));

vehicleData[index+3] = Convert.ToString(IntValidation(Console.ReadLine(), "Enter a realistic battery size value in kWh", 1, 110));

vehicleData[index + 4] = Console.ReadLine().ToUpper();

while (vehicleData[index + 4] != "P" && vehicleData[index + 4] != "D")

{

Console.WriteLine("Enter a suitable fuel type, P for petrol and D for diesel");

vehicleData[index + 4] = Console.ReadLine();

}

vehicleData[index+5] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic value for your vehicles range", 1, 500));

vehicleData[index+6] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic value for your vehicles range", 1, 1000));

}

if (vehicleType.ToUpper().Contains("EL"))

{

index = 3 + (engineDataLength \* engineCarAmount);

for (int i = dataLength + 3; i > index - electricDataLength; i--) //shifts everything forward in the array

{

vehicleData[i + electricDataLength] = vehicleData[i];

}

vehicleData[1] = Convert.ToString(Convert.ToInt32(vehicleData[1]) + 1);

Console.WriteLine("Type a name for your vehicle");

vehicleData[index] = Console.ReadLine();

Console.WriteLine("Enter the range of your vehicle");

vehicleData[index+1] = Convert.ToString(IntValidation(Console.ReadLine(), "Please enter a realistic value for your vehicles range", 1, 1000));

Console.WriteLine("Enter the battery size of your vehicle in kWh");

vehicleData[index+2] = Convert.ToString(IntValidation(Console.ReadLine(), "Enter a realistic battery size value in kWh", 1, 110));

}

File.WriteAllLines(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt", vehicleData);

LoadVehicleData();

}

**Replaces all the used text files with the restore copies**

public void RestoreToDefault()

{

File.Delete(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt");

File.Delete(@"E:\College Y2\Computer Science\Coursework\Settings.txt");

File.Copy(@"E:\College Y2\Computer Science\Coursework\Vehicle data restore.txt", @"E:\College Y2\Computer Science\Coursework\Vehicle data.txt");

File.Copy(@"E:\College Y2\Computer Science\Coursework\Settings restore.txt", @"E:\College Y2\Computer Science\Coursework\Settings.txt");

LoadSettings();

LoadVehicleData();

}

**Replaces the restore file to the current vehicleData.txt file**

public void UpdateRestore()

{

Console.WriteLine("Are youy sure you want to update the restore file, this cannot be undone? (Y / N)");

if (Console.ReadLine().ToUpper().Contains("Y"))

{

File.Delete(@"E:\College Y2\Computer Science\Coursework\Vehicle data restore.txt");

File.Copy(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt", @"E:\College Y2\Computer Science\Coursework\Vehicle data restore.txt");

Console.WriteLine("Your restore file has now been updated to support any vehicles added this session");

}

else

{

Console.WriteLine("Operation cancelled");

}

}

**Bubble sort to sort the vehicles in order of range**

public void SortVehicleData(int enCarAmount, int elCarAmount, int hyCarAmount) //bubble sort as there are too little values for a merge sort to be efficient, where 5, 3 and 7 are used, it is the length of the vehicle types data

{

string vehicleName = "";

if (SettingsData.GetVehicleType() == "ENGINE") //this is because after sorting vehicle data the indexes of the vehicles change so need to be able to update default vehicle with the updated index

{

vehicleName = EnCar[SettingsData.GetVehicleIndex()].GetName();

}

if (SettingsData.GetVehicleType() == "ELECTRIC")

{

vehicleName = ElCar[SettingsData.GetVehicleIndex()].GetName();

}

if (SettingsData.GetVehicleType() == "HYBRID")

{

vehicleName = HyCar[SettingsData.GetVehicleIndex()].GetName();

}

bool swap = true;

string[] temp = new string[7];

while(swap == true)

{

int count;

swap = false;

for (int index = 4; index < 3 + (enCarAmount - 1)\*5; index += 5)

{

if(Convert.ToInt32(vehicleData[index]) > Convert.ToInt32(vehicleData[index + 5])) //if the first vehicle range is larger than the second

{

count = index - 1;

for(int i = 0; i < 5; i++)

{

temp[i] = vehicleData[count]; //stores the 1st vehicle info starting with name hence -1 as the vehicle name is an index above range, then swaps them

vehicleData[count] = vehicleData[count + 5];

vehicleData[count + 5] = temp[i];

count++;

}

swap = true;

}

}

for (int index = 4 + enCarAmount \* 5; index < 4 + (enCarAmount \* 5) + ((elCarAmount -1)\* 3) - 1; index += 3) // -1 to use the name to start the switching opposed to range

{

if (Convert.ToInt32(vehicleData[index]) > Convert.ToInt32(vehicleData[index + 3]))

{

count = index - 1;

for (int i = 0; i < 3; i++)

{

temp[i] = vehicleData[count];

vehicleData[count] = vehicleData[count + 3];

vehicleData[count + 3] = temp[i];

count++;

}

}

}

for (int index = 4 + (enCarAmount \* 5) + (elCarAmount \* 3); index < 4 + (enCarAmount \* 5) + (elCarAmount \* 3) + ((hyCarAmount - 1) \* 7) - 1; index += 7)

{

if (Convert.ToInt32(vehicleData[index]) > Convert.ToInt32(vehicleData[index + 7]))

{

count = index - 1;

for (int i = 0; i < 7; i++)

{

temp[i] = vehicleData[count];

vehicleData[count] = vehicleData[count + 7];

vehicleData[count + 7] = temp[i];

count++;

}

}

}

}

File.WriteAllLines(@"E:\College Y2\Computer Science\Coursework\Vehicle data.txt", vehicleData);

LoadVehicleData();

if (SettingsData.GetVehicleType() == "ENGINE") //works out the default index from the name calculated before the sort

{

for(int i = 0; i < enCarAmount; i++)

{

if(EnCar[i].GetName() == vehicleName)

{

SettingsData.SetVehicleIndex(i);

}

}

}

if (SettingsData.GetVehicleType() == "ELECTRIC")

{

for (int i = 0; i < elCarAmount; i++)

{

if (ElCar[i].GetName() == vehicleName)

{

SettingsData.SetVehicleIndex(i);

}

}

}

if (SettingsData.GetVehicleType() == "HYBRID")

{

for (int i = 0; i < hyCarAmount; i++)

{

if (HyCar[i].GetName() == vehicleName)

{

SettingsData.SetVehicleIndex(i);

}

}

}

//saves vehicle data and settings, then reloads them

SaveSettings(SettingsData.GetWeatherIndex(), SettingsData.GetVehicleType(), SettingsData.GetVehicleIndex());

LoadSettings();

LoadVehicleData();

}

**Validates all user inputs that have to be converted into ints to be used in calculations**

public int IntValidation(string data, string exceptionErrorMSG, int min, int max)

{

bool valid = false;

int convertTo = -1;

while (valid != true)

{

try

{

convertTo = Convert.ToInt32(data);

if (convertTo <= max && convertTo >= min)

{

valid = true;

}

else

{

Console.ForegroundColor = ConsoleColor.DarkRed;

if(SettingsData.GetVehicleType() == "HYBRID" && max != 100)

{

Console.WriteLine(exceptionErrorMSG + ", the tank size for this vehicle is " + HyCar[SettingsData.GetVehicleIndex()].GetTankSize());

}

if (SettingsData.GetVehicleType() == "ENGINE")

{

Console.WriteLine(exceptionErrorMSG + ", the tank size for this vehicle is " + EnCar[SettingsData.GetVehicleIndex()].GetTankSize());

}

if (SettingsData.GetVehicleType() == "ELECTRIC")

{

Console.WriteLine(exceptionErrorMSG);

}

Console.ResetColor();

data = Console.ReadLine();

}

}

catch (Exception)

{

Console.ForegroundColor = ConsoleColor.DarkRed;

Console.WriteLine(exceptionErrorMSG);

Console.ResetColor();

data = Console.ReadLine();

}

}

return convertTo;

}

**class Cars**

protected string name;

protected double range;

public Cars(string name, double range)

{

this.name = name;

this.range = range;

}

public string GetName()

{

return name;

}

public double GetRange()

{

return range;

}

public void SetName(string name)

{

this.name = name;

}

public void SetRange(double range)

{

this.range = range;

}

**class ElectricCars : Cars**

private double batterySize;

public ElectricCars(string name, double range, double batterySize) : base(name, range)

{

this.batterySize = batterySize;

}

public double GetBatterySize()

{

return batterySize;

}

public void SetBatterySize(double batterySize)

{

this.batterySize = batterySize;

}

**Class EngineCars : Cars**

private double tankSize;

private double roadTax;

private char fuelType;

public EngineCars(string name, double range, double tankSize, double roadTax, char fuelType) : base(name, range)

{

this.tankSize = tankSize;

this.roadTax = roadTax;

this.fuelType = fuelType;

}

public double GetTankSize()

{

return tankSize;

}

public double GetRoadTax()

{

return roadTax;

}

public char GetFuelType()

{

return fuelType;

}

public void SetTankSize(double tankSize)

{

this.tankSize = tankSize;

}

public void SetRoadTax(double roadTax)

{

this.roadTax = roadTax;

}

public void SetFuelType(char fuelType)

{

this.fuelType = fuelType;

}

**Class HybridCars : Cars**

private double tankSize;

private double batterySize;

private char fuelType;

private double electricRange;

private double engineRange;

public HybridCars(string name, double range, double tankSize, double batterySize, char fuelType, double electricRange, double engineRange) : base(name, range)

{

this.tankSize = tankSize;

this.batterySize = batterySize;

this.fuelType = fuelType;

this.electricRange = electricRange;

this.engineRange = engineRange;

}

public double GetTankSize()

{

return tankSize;

}

public double GetBatterySize()

{

return batterySize;

}

public char GetFuelType()

{

return fuelType;

}

public double GetElectricRange()

{

return electricRange;

}

public double GetEngineRange()

{

return engineRange;

}

public void SetTankSize(double tankSize)

{

this.tankSize = tankSize;

}

public void SetBatterySize(double batterySize)

{

this.batterySize = batterySize;

}

public void SetFuelType(char fuelType)

{

this.fuelType = fuelType;

}

public void SetElectricRange(double electricRange)

{

this.electricRange = electricRange;

}

public void SetEngineRange(double engineRange)

{

this.engineRange = engineRange;

}

**Class Settings**

private int weatherIndex;

private string vehicleType;

private int vehicleIndex;

public Settings(int weatherIndex, string vehicleType, int vehicleIndex)

{

this.weatherIndex = weatherIndex;

this.vehicleType = vehicleType;

this.vehicleIndex = vehicleIndex;

}

public int GetWeatherIndex()

{

return weatherIndex;

}

public string GetVehicleType()

{

return vehicleType;

}

public int GetVehicleIndex()

{

return vehicleIndex;

}

public void SetWeatherIndex(int weatherIndex)

{

this.weatherIndex = weatherIndex;

}

public void SetVehicleType(string vehicleType)

{

this.vehicleType = vehicleType;

}

public void SetVehicleIndex(int vehicleIndex)

{

this.vehicleIndex = vehicleIndex;

}

**Class Weather**

private string name;

private bool on;

private double rangeDeduction;

public Weather(string name, bool on, double rangeDeduction)

{

this.name = name;

this.on = on;

this.rangeDeduction = rangeDeduction;

}

public string GetName()

{

return name;

}

public bool GetOn()

{

return on;

}

public double GetRangeDeduction()

{

return rangeDeduction;

}

public void SetOn(bool on)

{

this.on = on;

}

# Testing







# Improvements

Currently, the user can create a vehicle to simulate with no name at all which can create complications upon loading the vehicles as well as identifying the vehicle in the menu and statistics as it will just appear blank. To improve this, I could add input validation to make sure the string contains some character, so the user doesn’t accidentally create a vehicle with no name or intentionally. Another improvement I could make would be to have a dynamic electric price rather than a static, fixed value as electric prices have to potential to fluctuate which can create large inaccuracies during calculations. Adding a function to retrieve these prices would make my code larger and would only be valid for a particular electric provider unless I used averages; to add atop this, I could allow the user to select their electric provider and it would automatically get their prices or to allow the user to enter their own so it would be much more accurate with what they would realistically experience.