

Property Price Prediction

Alan Cahill

C00174533

Lecturer:

Dr. Lei Shi

Design Document

**Table of Contents**

Table of Contents

**Introduction1**

**Detailed Use Case4**

**System Sequence Diagram4**

**System Architecture4**

**Architecture Sequence of Execution4**

**Architecture Sequence During Development4**

**Database and Data Structure4**

**CSV File Data Formatting4**

**Pseudocode of Algorithm4**

# **Introduction**

The purpose of this document is to demonstrate the project form a design perspective. The document will detail the following:

* Detailed Use Case
* System Sequence Diagram
* System Architecture
* Architecture Sequence of Execution
* Architecture Sequence During Development
* Database and Data Structure
* CSV Files Data Formatting
* Pseudocode of Algorithm

**Detailed Use Case**

Detailed Use Case 1: Predict the price of a property

Actors: User

Brief Description: This use case begins when a user wishes to have a properties price predicted. The user will be prompted to enter in the details of the property. The entry of the details is achieved in stages. First the user will be asked to enter the county their property is in. If a county is entered incorrect the user will be prompted to re-enter the county data. On completion of entering the county data the user will proceed forward. The next step will prompt the user to enter in an address for the property. This address will be inputted and will then be geocoded using google maps geocoding API through python’s geocoder library. If the address cannot be found by google maps, then the user will be notified and asked to re-enter a more suitable address. When an address is finally found the user will continue to the next step. Here the user will be asked to enter in the number of bedrooms between 1 and 10 with the user being re-prompted if an invalid number is entered. On completion the user will then be prompted to enter in the number of bathrooms between 1 and 10 with the user being re-prompted if the number is invalid. Finally, the system will use the address to find out where the property situated in the areas created by the classification algorithm and will assign a label to the new property. The system will then check its historical records to determine if there is historical data for the combination entered by the user. If there is historical data, then the user will receive a prediction for the price of the property. If no historical data exists for that particular combination, then the user will be notified that the system cannot predict the price of the property.

Main success Scenario:

1. The use case begins when the user wants to have the price of a property predicted.
2. The user is prompted for the county the property resides in.
3. The user provides the county.
4. The system validates the entry.
5. The user is now prompted for the address.
6. The user enters the address.
7. The system validates the address via google maps.
8. The user is now prompted for the number of bedrooms in the property.
9. The user enters the number of bedrooms in the property.
10. The system validates the entry.
11. The user is prompted for the number of bathrooms in the property.
12. The user enters the number of bathrooms.
13. The system validates the number of bathrooms.
14. The system assigns an area label to the property using the classification algorithm.
15. The system will validate the data entered and constructed by checking its historical data to see if there exists data of the same combination as the data entered.
16. The system will extract the prediction from its historical data.
17. The system will display the price prediction to the user.

Alternatives:

4a1. The data entered is not valid.

4a2. The system notifies the user and returns to 2.

7a1. The address entered cannot be located by google maps.

7a2. The system notifies the user and returns to 5.

10a1. The data entered is not valid.

10a2. The system notifies the user and returns to 8.

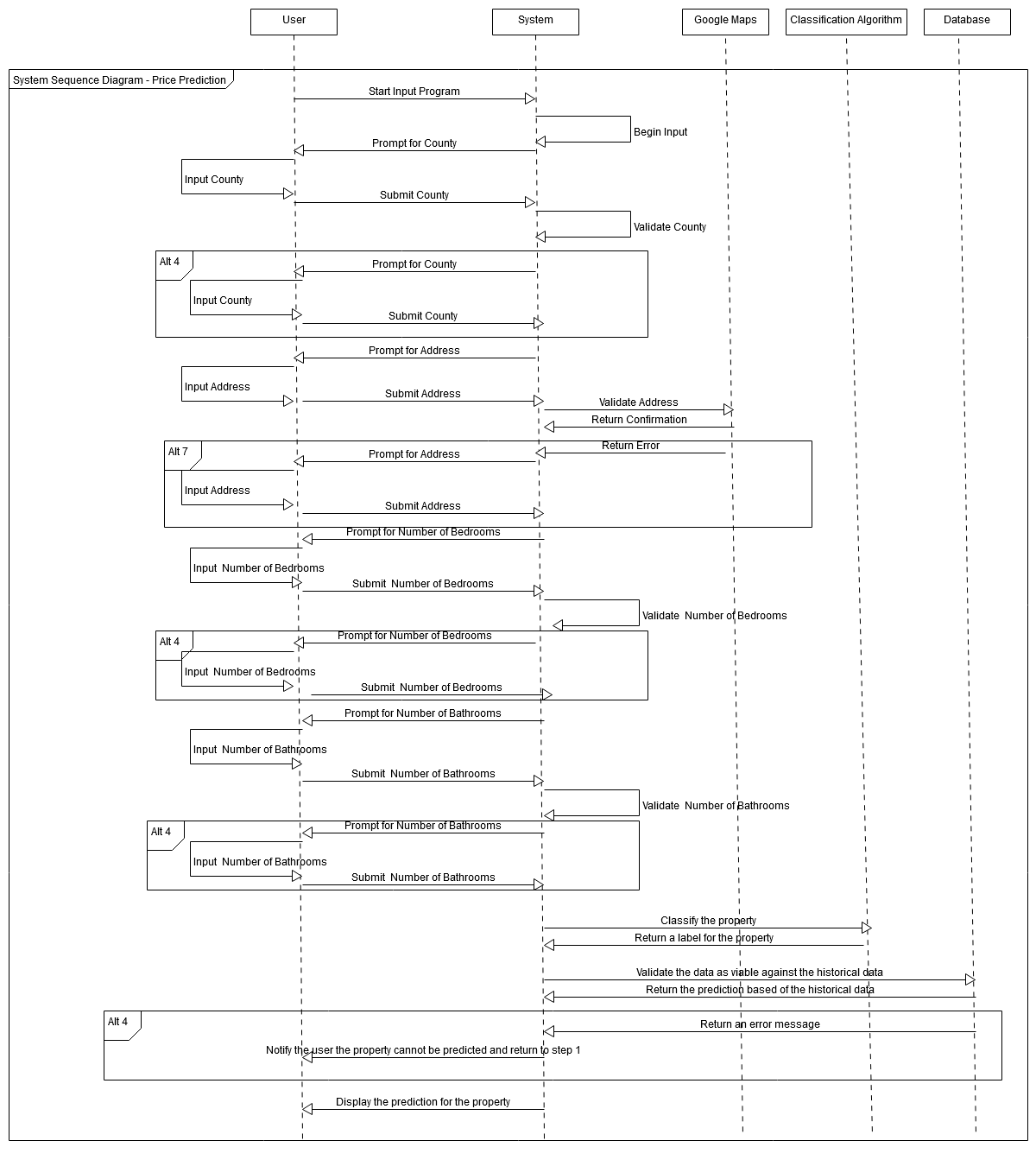
13a1. The data entered is not valid.

13a2. The system notifies the user and returns to 11.

15a1. The system has no correlating historical data.

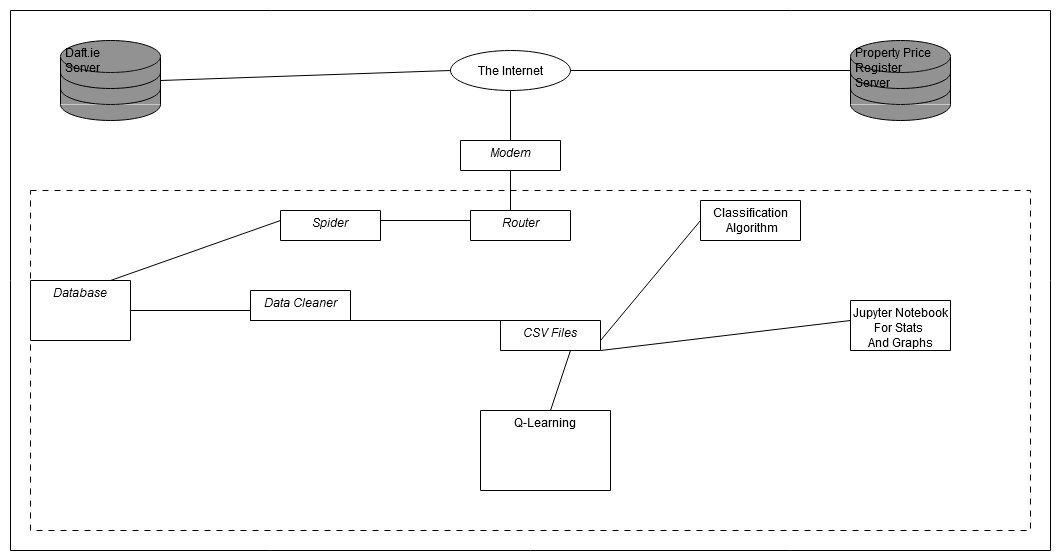
15a2. The system notifies the user that it cannot predict the price of the property.

15a3. The system returns to step 1.

**System Sequence Diagram**

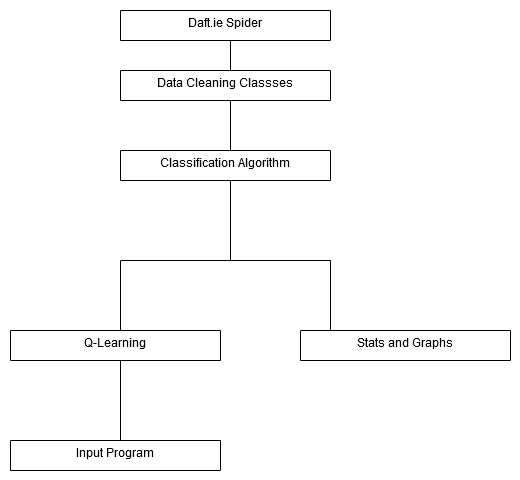
**System Architecture**

The System will be locally based but will utilise the servers of Daft.ie and the Property price register. The data set to be used in this project will be scraped from Daft.ie and combined with the data downloaded from the Property Price Register. On the local side and the side developed by this author the project will consist of a spider program for traversing and scraping the daft.ie website, A database, A data cleaning collection of python classes, CSV files for saving the cleaned data, A classification algorithm, a jupyter notebook displaying stats and graphs and a q-learning algorithm for making predictions. The architecture can be viewed below in fig 1.1



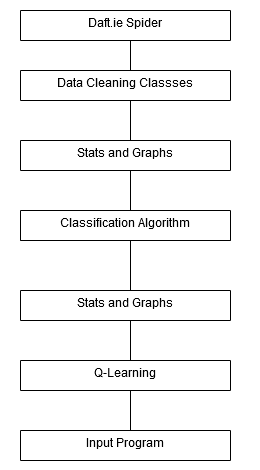
**Architecture sequence of execution:**

The sequence the architecture in this project will execute is displayed in Fig.3. As this project is designed for use on the console and is a primarily a backend mechanism it is essential that the users and readers understand how the system executes. The system begins by executing the spider which will scrape from daft.ie. The data cleaning classes will prepare the data for future use. This includes cleaning, integration and construction among other pre-processing techniques. The data cleaning classes use not only the daft data but also use the property price register data that was downloaded directly. The classification algorithm is used on the freshly prepared data and label is applied. With this data two possible processes can be carried out. The first is stats and graphs using a pre-prepared jupyter notebook and the second is Q-learning to create a model and make predictions. Finally the input program will be run to allow the users to interact with the system and achieve prediction sof their choice and interest.



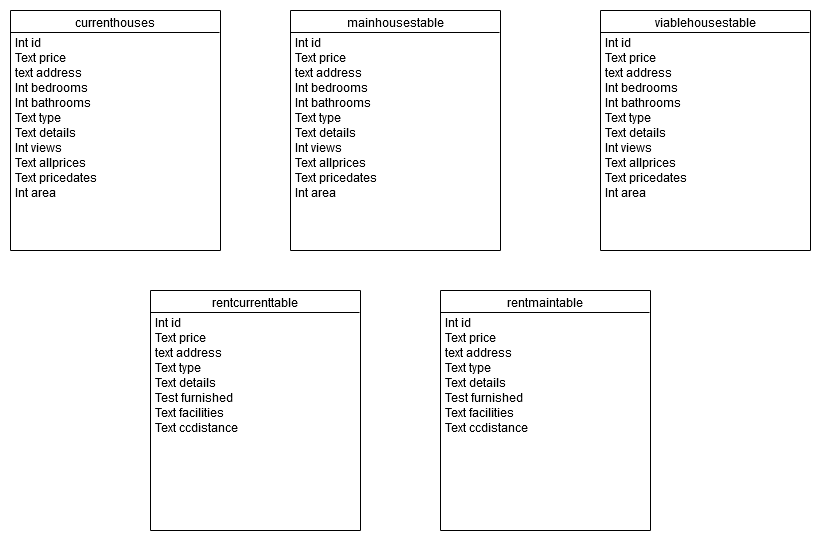
**Architecture Sequence During Development:**

The architecture during development differed slightly due to the use of statistics ad graphs to determine the necessary creation of a classifier algorithm. The classifier algorithm was not originally planned until the stats and graphs showed skewed results because of the location of the properties. As a results the classifier was created before a re-run of statistics was applied to confirm the data was ready for use in a machine learning algorithm. Below in Fig.4 is the sequence each piece of architecture was used when developing the project.



**Database and Data Structure**

The database for this project is extremely simple and does not even have any tables that interact with each other. Below is the entity relationship diagram for the database system implemented in the project. The currenthouses table saves the data directly after scraping. The main houses table contains every house detail that was scraped since the beginning of the project and the viable houses table contains all the houses without the duplicates. Rentcurrenttable contains the data straight after scraping and the mainrentable is the complete set of rent data downloaded since the beginning.



**CSV Files Data Formatting**

Once the data is cleaned it is saved to csv files. The data is automatically changed to string format when being saved in the csv files. Below details the structure of the csv files. The files are detailed based upon the group they belong to. For example, the Laois and Carlow csv files belong to the same group so detailing the structure of both is needless.

Allviablehousescsv:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite |

‘County’Houses:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite |

‘HouseType’Houses:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite |

‘Area’Houses:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite |

‘Area’HousesComplete:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite | LatLng | dateofsale | saleprice |

‘Area’Housetype’HousesComplete:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| price | address | bedrooms | bathrooms | type | area | Garage | Garden | Ensuite | LatLng | DateOfSale | SalePrice | label | Lattitude | Longitude |

‘Area’Housetype’BorderPoints:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Label | Compass | Section | Distance | Point | LatLng |

‘Area’HousType’CentrePoints:

|  |  |
| --- | --- |
| point | label |

Propregister:

|  |  |  |  |
| --- | --- | --- | --- |
| DateOfSale | address | price | LatLng |

**Pseudocode of Algorithm**

The pseudocode of the project is supplied below. There is a detailed description in English in the final report.

Daftspider.py

1. Import libraries
2. func run\_spider()
3. links = linkspider.crawl\_links()
4. connect to database
5. delete all entries in the database
6. for link in links
7. data = salesscraper.scrapedetail(link)
8. houseinfo = {}
9. assign relevant data to the house info dictionary
10. Commit the dictionary to the database
11. End loop

Linkspider.py

1. Crawl\_links()
2. Alllinks = []
3. Link = LinkGatherer.twentylinks(startinglink)
4. For I in link[0]:
5. Append I to list
6. While link[1]
7. Nextlink = link[1]
8. Link = LinkGatherer.twentylinks(nextlink)
9. For I in link[0]
10. Append I to list
11. End for
12. End while

LinkGatherer.py

1. Def twentylinks(link):
2. Create session
3. Set user agent
4. Html = session.get(link)
5. Html = html.txt
6. Soup = BeautifulSoup(html,’html5lib’)
7. Links = Soup.getalllinks()
8. Nextlink = Soup.getspecificLink
9. Return links, nextlink

Salesscraper.py

1. Def scrapedetails(link)
2. Create session
3. Add user agent
4. Retrieve html from link
5. Convert html to text
6. Convert text to bs4
7. Create the relevant strings
8. Assign the specified html tag contents to the specified string
9. Return the strings

DataCleaner.py

1. **func create\_csvs()**
2. Connect to database
3. Select all data in main houses table
4. Assign it to a dict
5. Call the appropriate cleaning/preprocessing function
6. Delete the details column
7. Write to a csv file
8. Func currency\_and\_strings(houses)
9. For t in houses
10. Change bedrooms,bathrooms and price columns to any integer within the associated string
11. Return houses
12. **Func removenovalues(houses)**
13. For house in houses
14. If the price,bedroom,batroom or area column is blank
15. Append that house to list (nothouse)
16. End for
17. For house in nothouse
18. Remove house from house
19. Return houses
20. **func change\_to\_int(houses)**
21. For house in houses
22. Change relavant column to relevant numeric variable
23. Return houses
24. **Func remove\_duplicates(houses)**
25. Dup = false
26. Newhouses = empty list
27. For house in houses
28. Inside the previous for loop -For I in newhouses
29. If the house is equal to i
30. Assign dup as true
31. If dup is false
32. Append the house to newhouses
33. Set dup false
34. End for
35. Return newhouses
36. Additional\_columns(houses):
37. For I in houses:
38. If the specified data is in the string
39. Set the relevant column to the specified
40. Return houses

Dublindatamaker.py

1. Def open\_propertyregister()
2. Open the csv file
3. Enter the data into the dicionary
4. Return the full dictionary
5. **Func geocode\_registerhouse()**
6. Open register data using function
7. For each register property
8. Geocode its address
9. Return the new dictionary
10. **Func compareGeocode(house):**
11. Open register data using function
12. For house in houses
13. For h in registerprice
14. If the two properties latitude and longitude are the same
15. Assign the new details to the house in houses
16. Return houses
17. **CreateDublin15():**
18. Open the csv
19. Assign it to a dictionary
20. Put all houses with Dublin 15 in them into a list
21. Wite them to a csv file
22. **Func googlecode(my\_list)**
23. For I in my\_list
24. Get geocodinh
25. End loop
26. Remove those without lat lang
27. Write to file

CSVSplitter.py

1. Read in the data
2. Loop all houses
3. Add the house to its county list
4. Write all lists to separate csv files

Tablescleaner.py

1. Read in from the database
2. Check for and remove duplicates
3. Save to new table

**Classifier**

Classifier\_Main Method.py

1. Import libraries
2. **Func open\_Corkcity**
3. Read in csv file
4. **Func open\_Dublin15()**
5. Red in csv file
6. **Func split\_housetype(all\_houses,htype)**
7. Return houses of passed house type
8. **Func split\_type\_detached(allhouses,htype1,htype2)**
9. Return houses of specified type
10. **Func Dublin\_outliers(newhouses)**
11. Remove all property outside the area
12. **Func cork\_outliers(newhouses)**
13. Remove all property outside the area
14. **Func run\_classifier()**
15. Covert latlng to a list instead of string
16. Ensure there are more than eight properties
17. Call the number \_of\_areas function
18. Check for border instance as confirmation there is an area
19. Write the relevant data to a csv file
20. Run for cork city
21. Write relevant data to file
22. Run for dublin
23. Write relevant data to file

Point\_setter.py

1. **Func check\_coverage(coverage houses)**
2. Get countof coverage houses with label
3. Divide into the total
4. Multiply by 100 for percentage
5. Return the percentage
6. **Func set\_point()**
7. Loop the houses
8. If the house has a label it is excluded
9. Otherwise if price is within range
10. Assign as point
11. Return point
12. **Test\_area()**
13. Ensure the area has 8% of the total houses
14. If it doesn’t
15. Reset all labels equal to the test label
16. Return
17. **Ninety\_percent\_runner()**
18. call percentage checker function
19. while the percentage is less than 85 percent
20. while the area test is false
21. set the point
22. if the point is valid
23. while the gate is a certain value and the gatetest is false
24. call the section splitter
25. test the area
26. set percentage and dictionary variables for returning
27. return everything once all loops are completed
28. **check number()**
29. ensure the areas is between 3 and 11
30. return Boolean
31. **opening\_gate()**
32. set point
33. loop through sectionsplitter until a viable point is found
34. **number of areas()**
35. loop through the gate level by 5,000 until a proper return is found
36. then return everything

section\_splitting\_run program.py

1. **split the points into compass position lists**
2. send each compass list to the step section function
3. reassign the return to a complete list
4. return everything

SectionStepper.py

1. **orderpoints()**
2. calculate the distance of each point to the centre point
3. order based off distance-ascending
4. **classify\_points\_section()**
5. depending what compass position the pooint
6. if the point is on the subsection line or to the left/right then it is within that subsection
7. **step\_through\_nsew\_points**
8. for each point
9. if the point is within the specified distance
10. within the price gate
11. then it is labelled the same
12. otherwise
13. check the count
14. if it is 1 then continue
15. if it is 2 then the border/end point can be set here
16. **step through points**
17. for each point
18. if the point is within the specified distance
19. within the price gate
20. then it is labelled the same
21. otherwise
22. check the count
23. if it is 1 then continue
24. if it is 2 then the border/end point can be set here
25. **step\_section()**
26. call all the necessary functions
27. **step nsew**
28. call the necessary functions

SectionSplitter.py

1. Plot the compass subsection lines
2. Iterate 9degrees each time
3. Using graph theory formula
4. Y = mx -b

Compass splitter

1. Assign a compass postion
2. Based off of the relative position to the centre point

Q-Learning

Initialise\_q\_environment.py

1. **Createdict()**
2. Initialse the q enivironment to a 4d linked list usint dictionaires and lists
3. **Set exit points**
4. Loop the environment and set and exit point
5. At the specified point

Actor.py

1. **Get predictions()**
2. Loop the environment until an exit is found
3. Ensure the loop does not loop back on itself
4. Ensure larger loop is not at play
5. Set a maximum step count
6. **Update all q values**
7. Step through the environment
8. Exit on exit
9. Repeat until no further q value updates are possible
10. **Enter environment**
11. Create environment
12. Entre exit points
13. Keep looping until no more updates possible

Run Q learning.py

1. Enter the environment for each segment
2. Then get predictions for each segment
3. Write to file

Data Segmentation.

1. **Split\_data()**
2. Split the data into an 80 twenty split
3. **Create a segment()**
4. For each of combination(1-11,1-11,1-11)
5. Create an environment
6. **Segment the data()**
7. For each segment
8. Append any relevant properties to the list
9. Delete any segment without a property appended
10. **Import then segment**
11. Import the csv file
12. Call relevant functions

Q value handler.py

1. Implement the q learning formala

Step chooser.py

1. Choose the property with the highest q value associated with it
2. If equal values exist
3. Choose at random
4. **Get max**
5. Find the maximum of the suppled q values