London Tube Navigation Expert System in CLIPS

PART - 3

<u>Team 4</u>

Chadha, Harshita

Sanghvi, Prima

Tadikonda, Naga Sai Bhavya

Department of Computer Science

George Washington University

Professor Stephen Kaisler

Department of Computer Science

George Washington University

Submitted on - **December 16, 2023**

TABLE OF CONTENTS

1. Introduction

This report serves to provide an overview of part 3 of the development of a London Tube Navigation Expert System that provides travelers information about the complex London underground network. Our goals in this part of project development were to modify the expert system to incorporate changes according to provided comments after part-2 submission as well as to modify the system to deal with station closures (the main goal for part-3). Further, we also worked on enhancing our option-based interface to allows users to interact with the expert system more easily and use the information contained in the knowledge base to plan their next exciting outcome in London!

This report is made up of eight main sections. The first and present section serves as an introduction to the development cycle in part-3 and provides a brief overview of the work done. Section 2 of this document summarizes the steps that are to be followed to initialize and run the expert system. Sections 4, 5, and 6 detail the templates, facts, rules, and functions that work together to make the system function. Lastly, conclusions and references for all external sources are provided in sections 7 through 8.

2. Instructions for Running the Expert System

In the file exchange for group 4 on blackboard, 2 items have been submitted. The first is a zip folder titled “Part3_Group4_System.zip”. This folder contains separate txt source files for rules, functions, facts, templates, and loading. Within this folder, the file "LoadingFiles.txt" contains a loader function, (loading), that loads all the remaining files and initializes the system. The second submitted item is the present document titled “Part3_Group4_Report.docx”.

To run the system, the following steps should be followed -

1. Download the and unzip the folder “Part3_Group4_System.zip” from Group-4’s file exchange on blackboard.

2. In the CLIPS console, switch the directory to the unzipped folder “Part3_Group4_System”.

3. In the CLIPS console enter the following command -

```
(load "LoadingFiles.txt")
```

4.In the CLIPS console enter the following command to invoke the main loader function.

```
(loading)
```

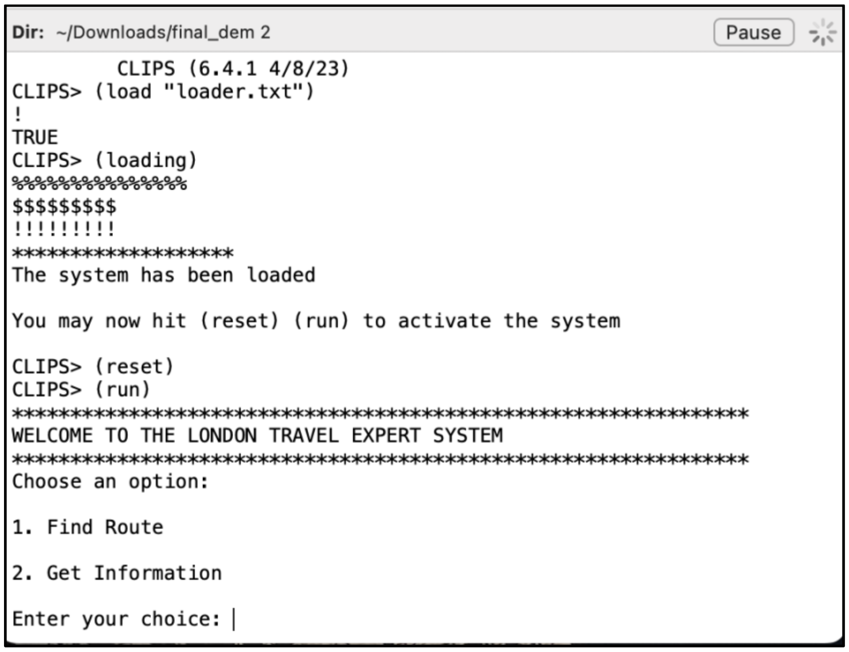
5.Enter the following commands in the CLIPS Console

```
(reset)
(run)
```

Following the above steps initializes the system and a menu driven, prompt like expert system is loaded which can be interacted with to plan trips, get more information on attractions in London, etc.

3. Expert System Usage Examples

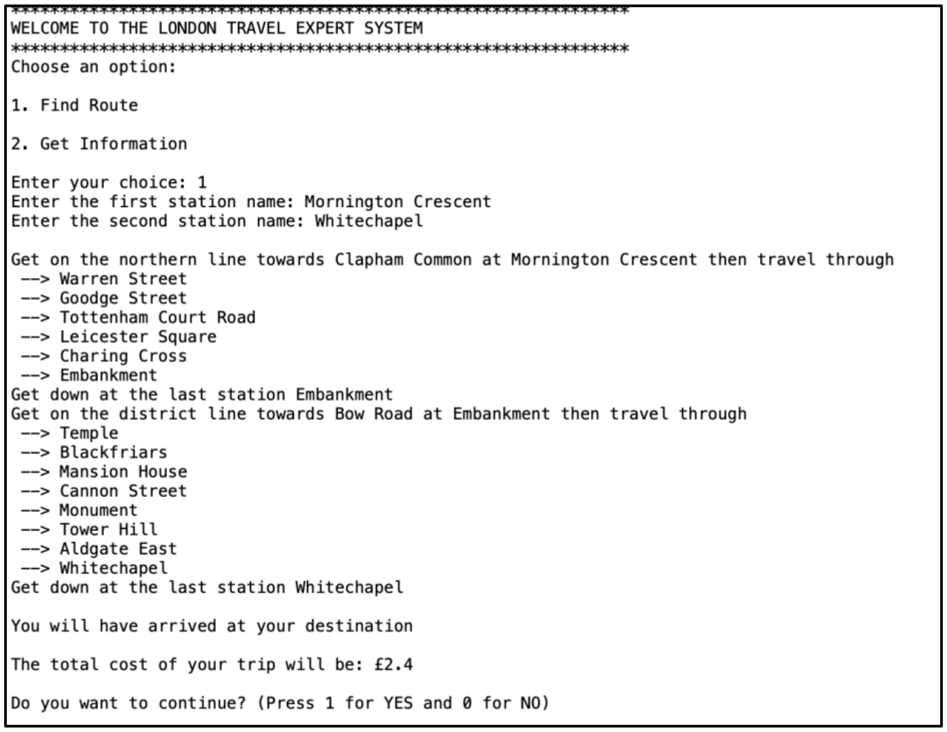
In this section we provide an overview of the capabilities of our London Tube Navigation System and demonstrate some examples of usage. When the commands illustrated in section 2 above are executed, the system is initialized and ready for interaction. This initial interface is shown in the figure below -



```
         CLIPS (6.4.1 4/8/23)
CLIPS> (load "loader.txt")
!
TRUE
CLIPS> (loading)
%%%%%%%%%%%%%%%
$$$$$$$$$
!!!!!!!!!
*******************
The system has been loaded

You may now hit (reset) (run) to activate the system

CLIPS> (reset)
CLIPS> (run)
***************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
***************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: |
```

Figure 1: Initial User Interface for The London Tube Expert System

The initial interface offers the user two choices - Find Route and Get Information. When the first option, Find Route, is selected, the user is prompted to enter the name of the start and the ending station. This is shown in the figure below -

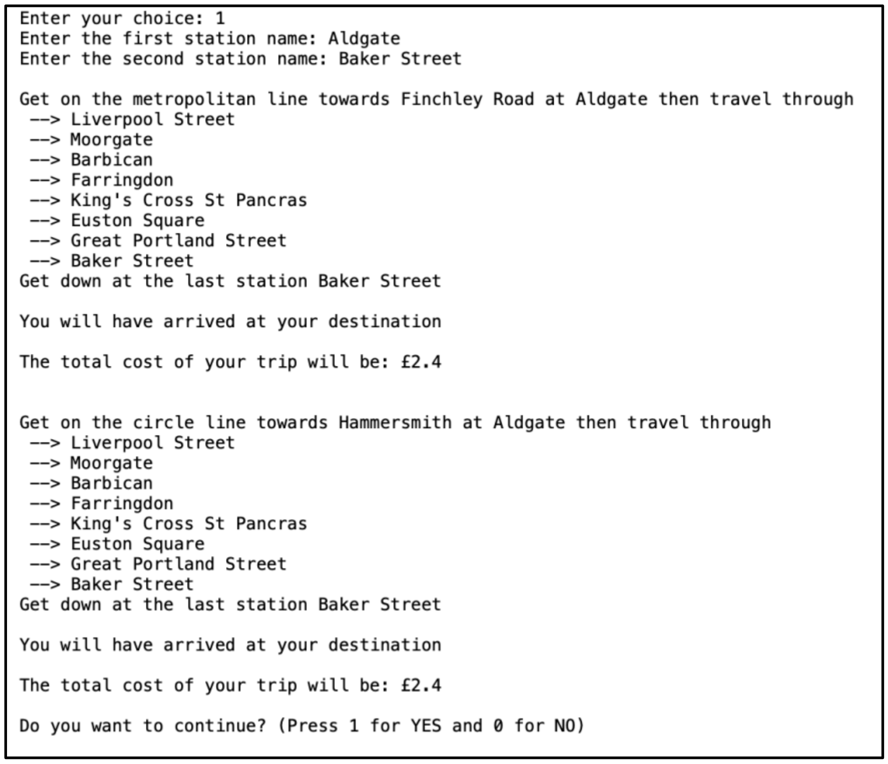


```
****************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
****************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 1
Enter the first station name: Mornington Crescent
Enter the second station name: Whitechapel

Get on the northern line towards Clapham Common at Mornington Crescent then travel through
 --> Warren Street
 --> Goodge Street
 --> Tottenham Court Road
 --> Leicester Square
 --> Charing Cross
 --> Embankment
Get down at the last station Embankment
Get on the district line towards Bow Road at Embankment then travel through
 --> Temple
 --> Blackfriars
 --> Mansion House
 --> Cannon Street
 --> Monument
 --> Tower Hill
 --> Aldgate East
 --> Whitechapel
Get down at the last station Whitechapel

You will have arrived at your destination

The total cost of your trip will be: £2.4

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 2: Calculating Route Using the Expert System

As can be seen, entering the name of the stations generates the route for the trip along with the cost of making the trip. To calculate the route, a dynamic fact is initialized to capture the user input. Following this, if the route to be calculated happens to be for stations on the same line, a different rule executes simply outputting the path.

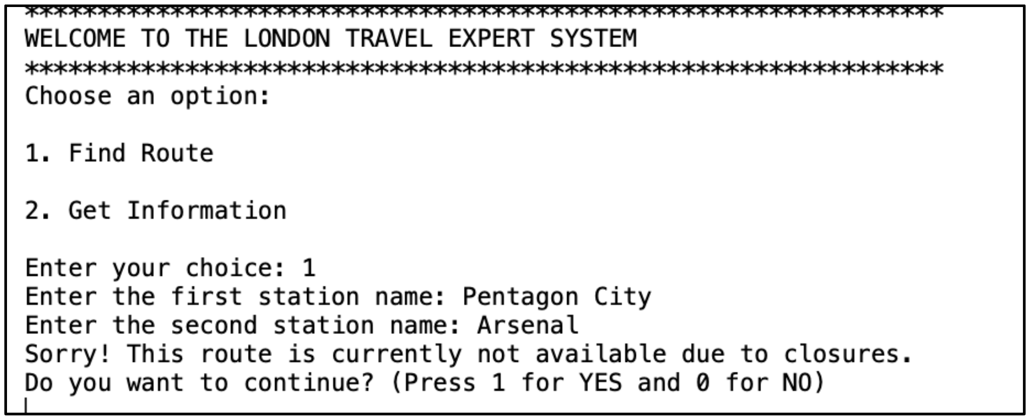
If in case the stations happen to be on different lines, a separate rule executes and uses a semi-bfs (breadth first search) based approach to obtain a set of possible routes. Following this, based on some optimality parameters routes are filtered to obtain the ones with the minimal number of switches and intermediate stations involved. The route calculation methodology is defined in further detail in section 4 below. If there exist multiple optimal routes of travel for two entered stations, all of them are output after filtering rules are executed. This is shown in the case of the below illustrated example -



```
Enter your choice: 1
Enter the first station name: Aldgate
Enter the second station name: Baker Street

Get on the metropolitan line towards Finchley Road at Aldgate then travel through
 --> Liverpool Street
 --> Moorgate
 --> Barbican
 --> Farringdon
 --> King's Cross St Pancras
 --> Euston Square
 --> Great Portland Street
 --> Baker Street
Get down at the last station Baker Street

You will have arrived at your destination

The total cost of your trip will be: £2.4


Get on the circle line towards Hammersmith at Aldgate then travel through
 --> Liverpool Street
 --> Moorgate
 --> Barbican
 --> Farringdon
 --> King's Cross St Pancras
 --> Euston Square
 --> Great Portland Street
 --> Baker Street
Get down at the last station Baker Street

You will have arrived at your destination

The total cost of your trip will be: £2.4

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 3: Existence Of Multiple Optimal Routes

In case, the user enters stations that do not exist on the metro line or in case the user makes a spelling mistake, an error rule runs, and a message is displayed. This is shown below -



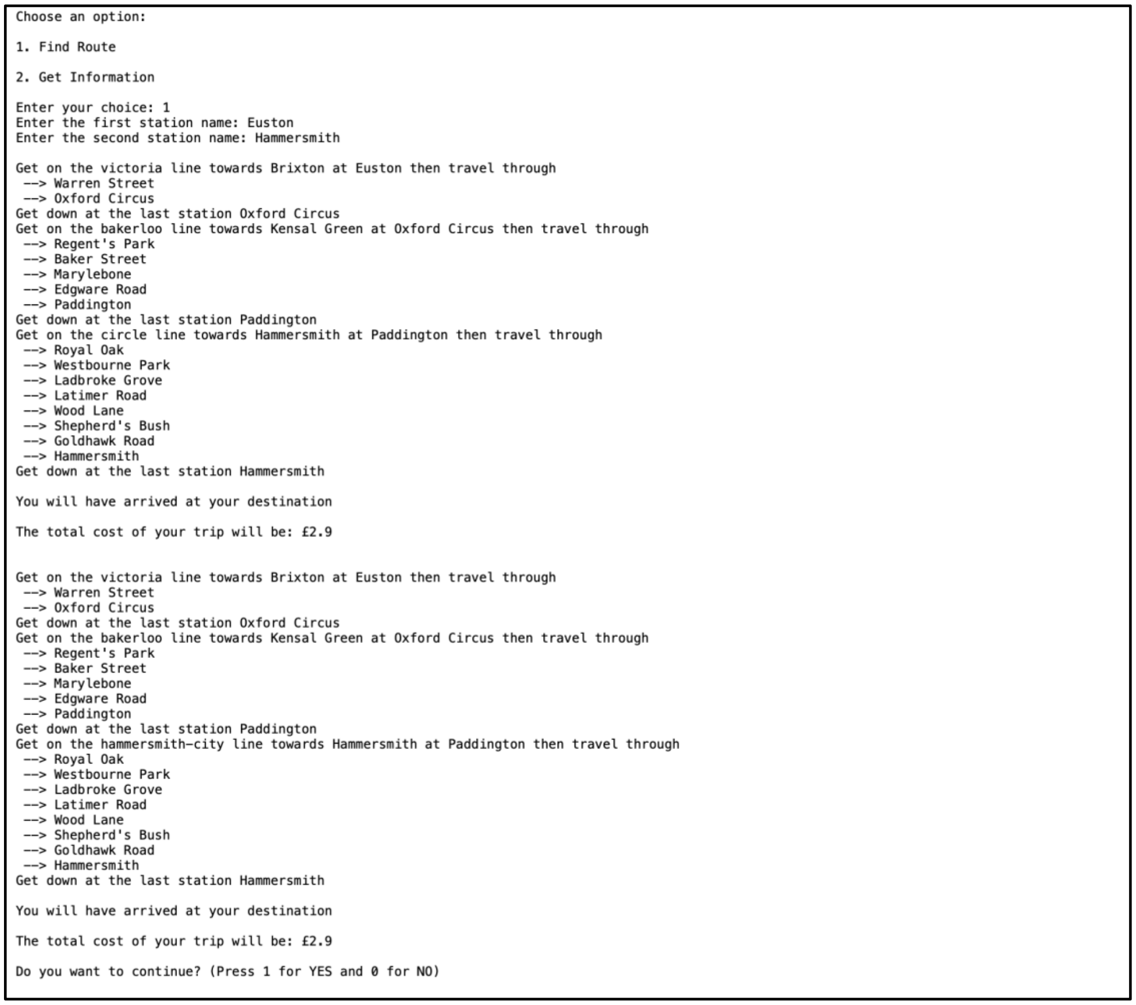
```
***************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
***************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 1
Enter the first station name: Pentagon City
Enter the second station name: Arsenal
Sorry! This route is currently not available due to closures.
Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 4: Error Message For Spelling Errors And Non-Existent Stations

Station Closure

For part 3, some stations have closure and, in that case, the expert system gives alternate routes. Below are few examples of routes with closed stations:

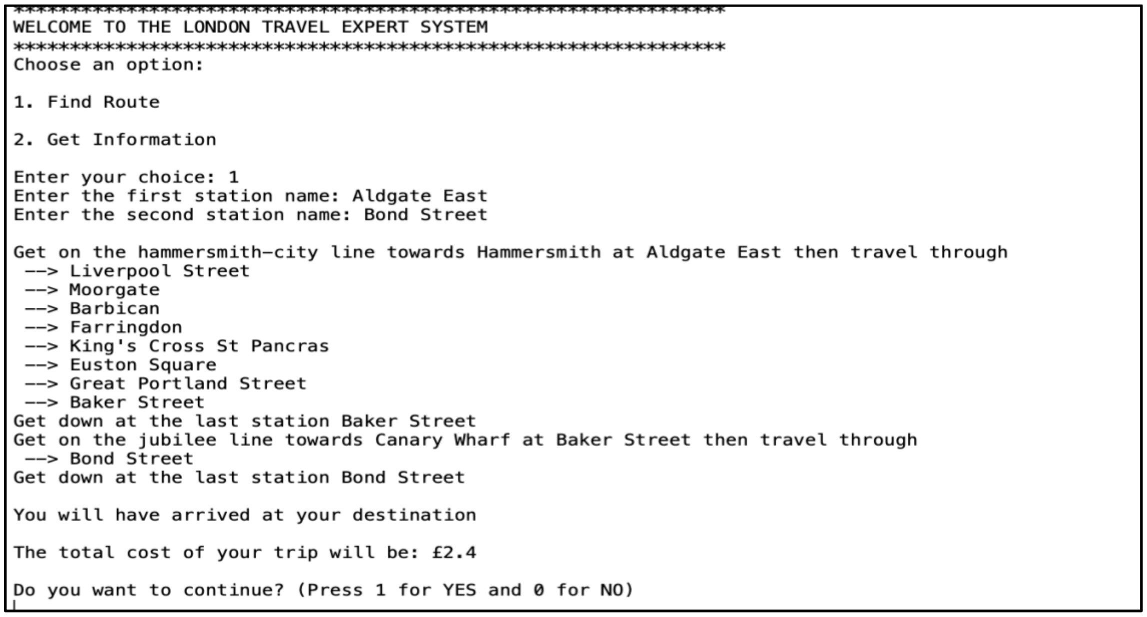
- **Euston to Hammersmith**



```
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 1
Enter the first station name: Euston
Enter the second station name: Hammersmith

Get on the victoria line towards Brixton at Euston then travel through
 --> Warren Street
 --> Oxford Circus
Get down at the last station Oxford Circus
Get on the bakerloo line towards Kensal Green at Oxford Circus then travel through
 --> Regent's Park
 --> Baker Street
 --> Marylebone
 --> Edgware Road
 --> Paddington
Get down at the last station Paddington
Get on the circle line towards Hammersmith at Paddington then travel through
 --> Royal Oak
 --> Westbourne Park
 --> Ladbroke Grove
 --> Latimer Road
 --> Wood Lane
 --> Shepherd's Bush
 --> Goldhawk Road
 --> Hammersmith
Get down at the last station Hammersmith

You will have arrived at your destination

The total cost of your trip will be: £2.9


Get on the victoria line towards Brixton at Euston then travel through
 --> Warren Street
 --> Oxford Circus
Get down at the last station Oxford Circus
Get on the bakerloo line towards Kensal Green at Oxford Circus then travel through
 --> Regent's Park
 --> Baker Street
 --> Marylebone
 --> Edgware Road
 --> Paddington
Get down at the last station Paddington
Get on the hammersmith-city line towards Hammersmith at Paddington then travel through
 --> Royal Oak
 --> Westbourne Park
 --> Ladbroke Grove
 --> Latimer Road
 --> Wood Lane
 --> Shepherd's Bush
 --> Goldhawk Road
 --> Hammersmith
Get down at the last station Hammersmith

You will have arrived at your destination

The total cost of your trip will be: £2.9

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 5: Route options for Euston to Hammersmith

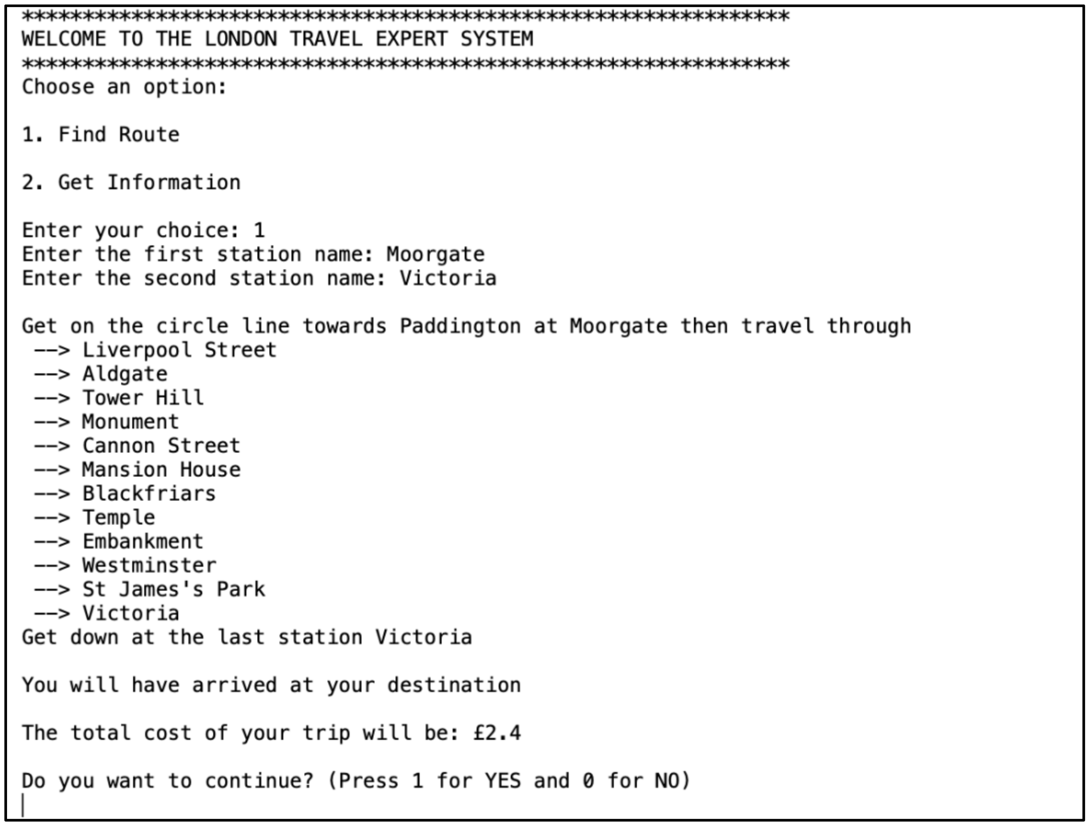
- **Aldgate East to Bond Street**



```
****************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
****************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 1
Enter the first station name: Aldgate East
Enter the second station name: Bond Street

Get on the hammersmith-city line towards Hammersmith at Aldgate East then travel through
 --> Liverpool Street
 --> Moorgate
 --> Barbican
 --> Farringdon
 --> King's Cross St Pancras
 --> Euston Square
 --> Great Portland Street
 --> Baker Street
Get down at the last station Baker Street
Get on the jubilee line towards Canary Wharf at Baker Street then travel through
 --> Bond Street
Get down at the last station Bond Street

You will have arrived at your destination

The total cost of your trip will be: £2.4

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 6: Route options for Aldgate East to Bond Street

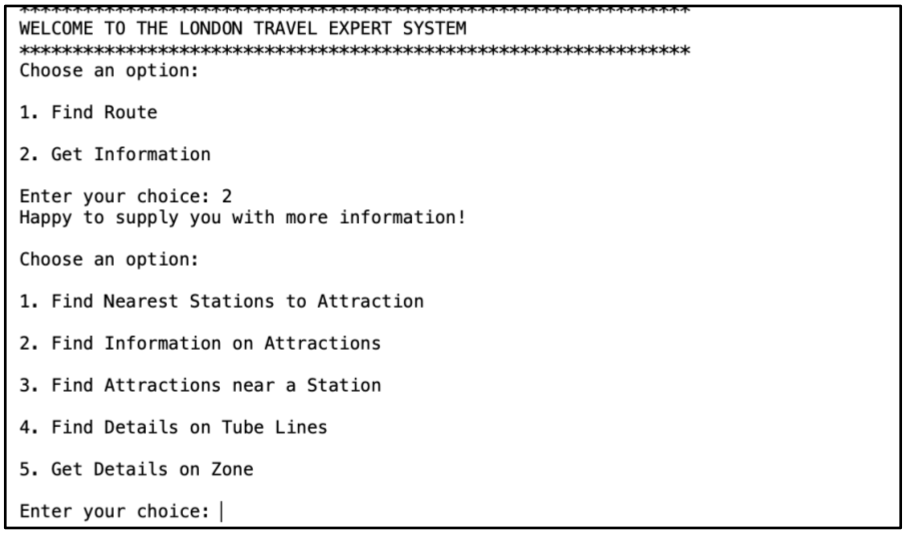
- **Moorgate to Victoria**



```
****************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
****************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 1
Enter the first station name: Moorgate
Enter the second station name: Victoria

Get on the circle line towards Paddington at Moorgate then travel through
 --> Liverpool Street
 --> Aldgate
 --> Tower Hill
 --> Monument
 --> Cannon Street
 --> Mansion House
 --> Blackfriars
 --> Temple
 --> Embankment
 --> Westminster
 --> St James's Park
 --> Victoria
Get down at the last station Victoria

You will have arrived at your destination

The total cost of your trip will be: £2.4

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 7: Route options for Moorgate to Victoria

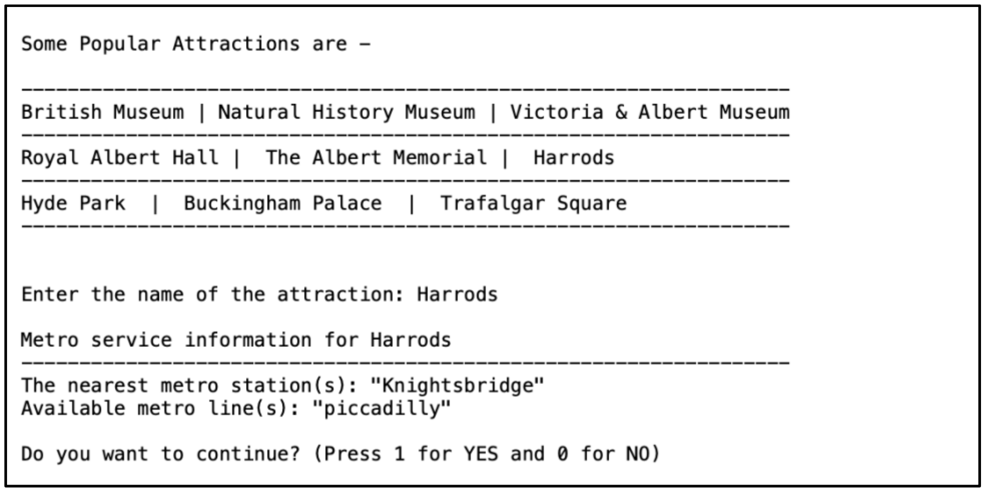
Next, in case the user does not want to calculate the route but wants information on attractions, line details, zone details or nearest station near some attraction in and around London itself, the second option, Get Information, can be used. Selecting this option produces another menu which shows the user the different types of actions they can perform to get information from the system. This is shown in the figure below -



```
**************************************************************
WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM
**************************************************************
Choose an option:

1. Find Route

2. Get Information

Enter your choice: 2
Happy to supply you with more information!

Choose an option:

1. Find Nearest Stations to Attraction

2. Find Information on Attractions

3. Find Attractions near a Station

4. Find Details on Tube Lines

5. Get Details on Zone

Enter your choice: |
```

Figure 8: Option Two - Finding Information on Attractions

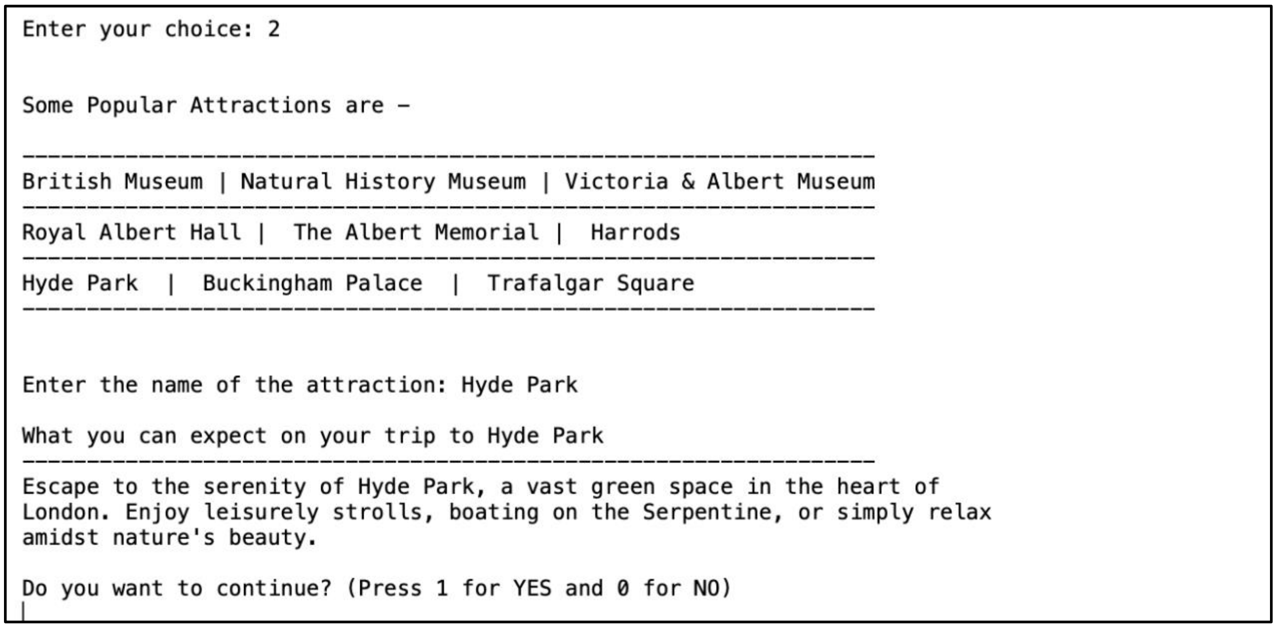
The first sub-option, when selected, shows the list of few famous attractions to the user, and prompts to enter the name of an attraction and based on fact matches according to defined rules, outputs the nearest metro stations to the attraction and the lines on which these metro stations are. This is shown in the example in the figure below-



```
Some Popular Attractions are -

-----------------------------------------------------------------
British Museum | Natural History Museum | Victoria & Albert Museum
-----------------------------------------------------------------
Royal Albert Hall |  The Albert Memorial |  Harrods
-----------------------------------------------------------------
Hyde Park  |  Buckingham Palace  |  Trafalgar Square
-----------------------------------------------------------------


Enter the name of the attraction: Harrods

Metro service information for Harrods
-----------------------------------------------------------------
The nearest metro station(s): "Knightsbridge"
Available metro line(s): "piccadilly"

Do you want to continue? (Press 1 for YES and 0 for NO)
```

Figure 9: Getting Location Information about London Attractions

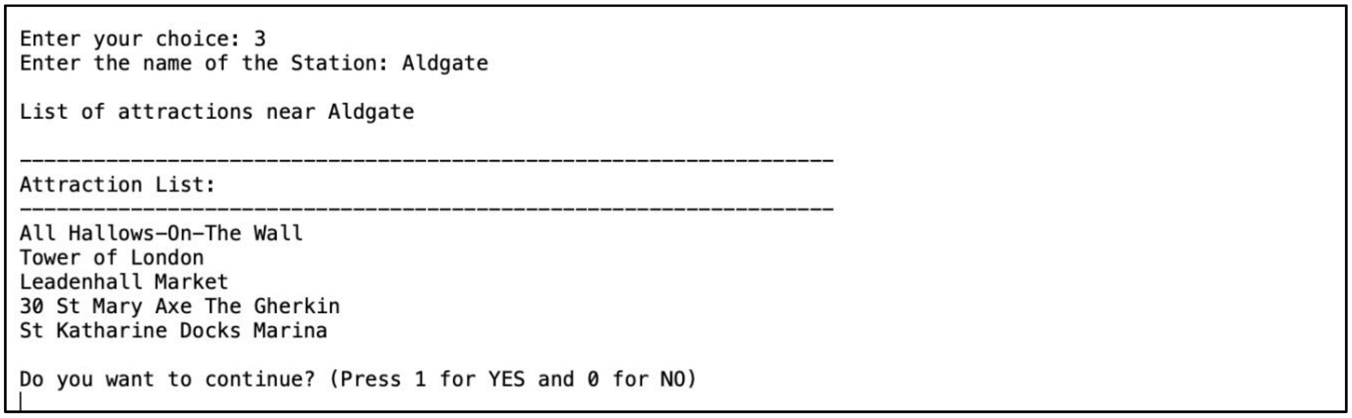
The second option of the get information sub-menu also shows the list of few famous attractions to the user and prompts the user to enter the name of the attraction and outputs a brief description of the entered attraction. This is shown in the figure below -



```
Enter your choice: 2


Some Popular Attractions are -

-----------------------------------------------------------------
British Museum | Natural History Museum | Victoria & Albert Museum
-----------------------------------------------------------------
Royal Albert Hall |  The Albert Memorial |  Harrods
-----------------------------------------------------------------
Hyde Park  |  Buckingham Palace  |  Trafalgar Square
-----------------------------------------------------------------


Enter the name of the attraction: Hyde Park

What you can expect on your trip to Hyde Park
-----------------------------------------------------------------
Escape to the serenity of Hyde Park, a vast green space in the heart of
London. Enjoy leisurely strolls, boating on the Serpentine, or simply relax
amidst nature's beauty.

Do you want to continue? (Press 1 for YES and 0 for NO)
|
```

Figure 10: Getting Attraction's Description from Name

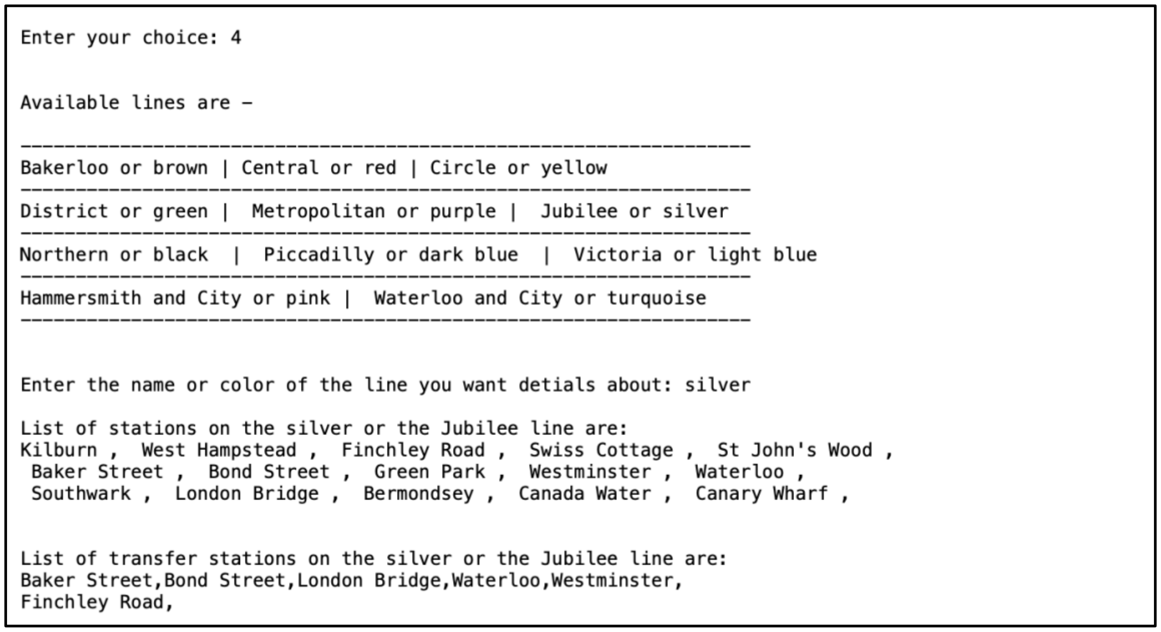
The third option, in the sub-menu allows the user to enter the name of a train station on the London Tube System and in return outputs a list of all attractions near this station in a list format arranged in increasing order of distance within an acceptable radius. An example of this is shown in the figure below -



```
Enter your choice: 3
Enter the name of the Station: Aldgate

List of attractions near Aldgate

-----------------------------------------------------------------
Attraction List:
-----------------------------------------------------------------
All Hallows-On-The Wall
Tower of London
Leadenhall Market
30 St Mary Axe The Gherkin
St Katharine Docks Marina

Do you want to continue? (Press 1 for YES and 0 for NO)
|
```

Figure 11: Getting Nearest Attractions to a Given Station

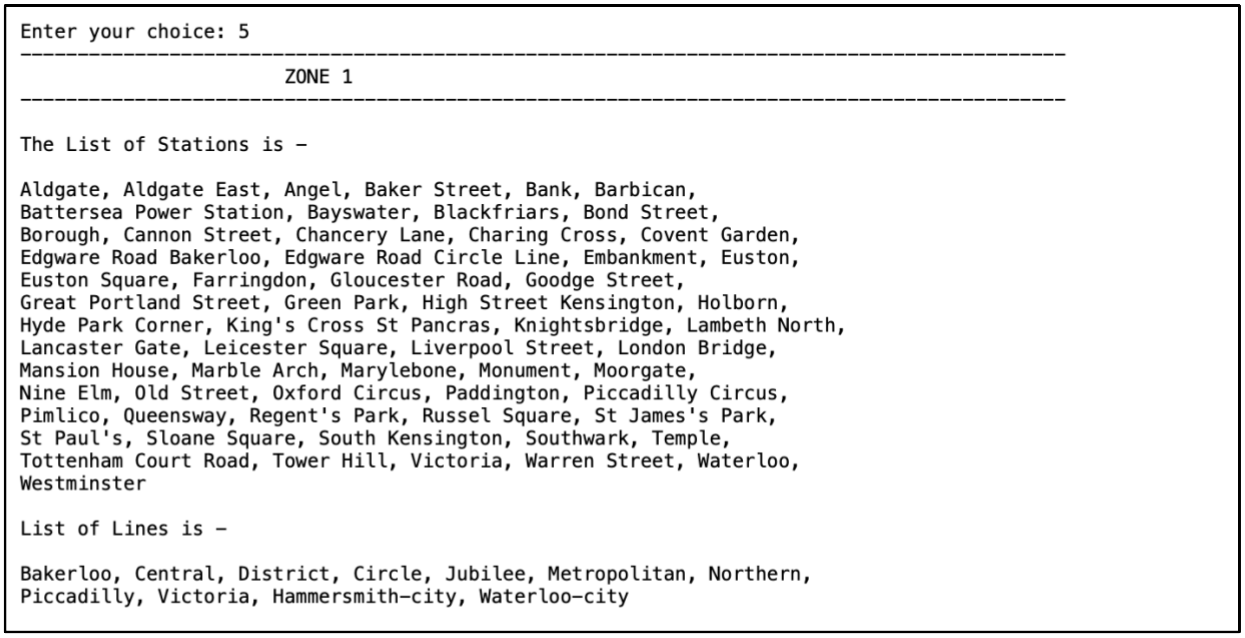
The fourth option, in the sub-menu, shows the user the list of available lines and allows the user to enter the line name or line color they want information on and in return outputs a list of all stations on that specific line and the list of all transfer stations on that line. An example of this is shown in the figure below -



```
Enter your choice: 4


Available lines are -

-----------------------------------------------------------------
Bakerloo or brown | Central or red | Circle or yellow
-----------------------------------------------------------------
District or green |  Metropolitan or purple |  Jubilee or silver
-----------------------------------------------------------------
Northern or black  |  Piccadilly or dark blue  |  Victoria or light blue
-----------------------------------------------------------------
Hammersmith and City or pink |  Waterloo and City or turquoise
-----------------------------------------------------------------


Enter the name or color of the line you want detials about: silver

List of stations on the silver or the Jubilee line are:
Kilburn ,  West Hampstead ,  Finchley Road ,  Swiss Cottage ,  St John's Wood ,
 Baker Street ,  Bond Street ,  Green Park ,  Westminster ,  Waterloo ,
 Southwark ,  London Bridge ,  Bermondsey ,  Canada Water ,  Canary Wharf ,


List of transfer stations on the silver or the Jubilee line are:
Baker Street,Bond Street,London Bridge,Waterloo,Westminster,
Finchley Road,
```

Figure 12: Getting Station information on Given Lines

Finally, when the user selects the fifth option it displays the list of all stations and lines in zone 1, zone 2 and zone 1/2. An example of this is shown in the figure below -



```
Enter your choice: 5
-------------------------------------------------------------------------------------------
                       ZONE 1
-------------------------------------------------------------------------------------------

The List of Stations is -

Aldgate, Aldgate East, Angel, Baker Street, Bank, Barbican,
Battersea Power Station, Bayswater, Blackfriars, Bond Street,
Borough, Cannon Street, Chancery Lane, Charing Cross, Covent Garden,
Edgware Road Bakerloo, Edgware Road Circle Line, Embankment, Euston,
Euston Square, Farringdon, Gloucester Road, Goodge Street,
Great Portland Street, Green Park, High Street Kensington, Holborn,
Hyde Park Corner, King's Cross St Pancras, Knightsbridge, Lambeth North,
Lancaster Gate, Leicester Square, Liverpool Street, London Bridge,
Mansion House, Marble Arch, Marylebone, Monument, Moorgate,
Nine Elm, Old Street, Oxford Circus, Paddington, Piccadilly Circus,
Pimlico, Queensway, Regent's Park, Russel Square, St James's Park,
St Paul's, Sloane Square, South Kensington, Southwark, Temple,
Tottenham Court Road, Tower Hill, Victoria, Warren Street, Waterloo,
Westminster

List of Lines is -

Bakerloo, Central, District, Circle, Jubilee, Metropolitan, Northern,
Piccadilly, Victoria, Hammersmith-city, Waterloo-city
```

Figure 13: Getting station and line information on zone.

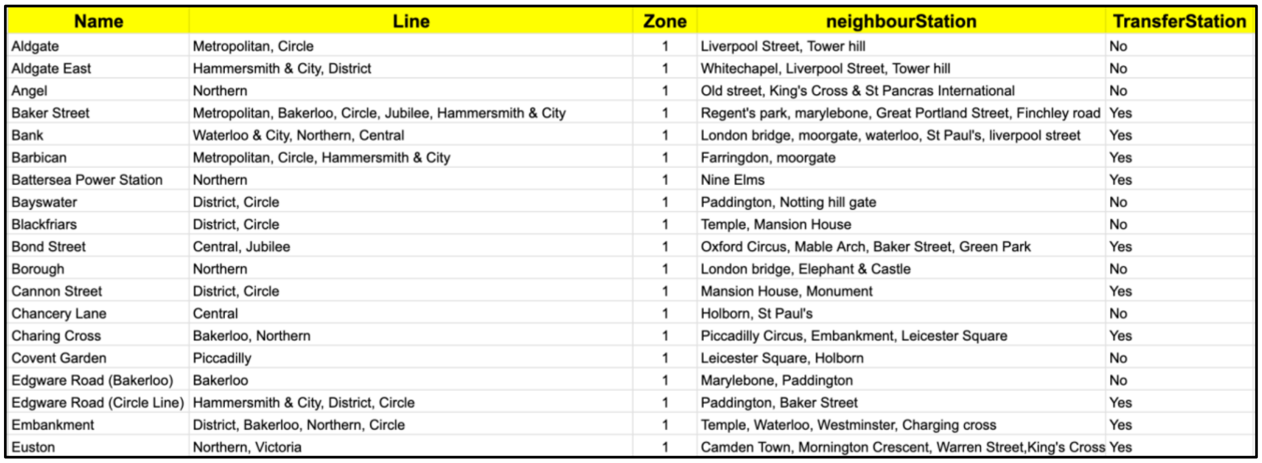
Thus, the created London Tube navigation system offers a menu driven user interface that allows the user to interact with the knowledge base of the expert system and obtain useful functionality from it. While this section served to provide examples of usage, in the following sections we describe the process of collecting data and crafting facts, rules and functions to obtain the final system.

4. Facts, Templates and Data Collection

4.1. Data Collection

This section serves to briefly describe the identified entities that would form the knowledge base for our expert system and the data collection process to create facts on top of these entities. After careful and detailed analysis of the underground London tube system in the phase one of exert system development, we identified 4 key entity categories to help us model our expert system and perform key operations such as route and fare calculations. These were - station, lines, attractions and fares.

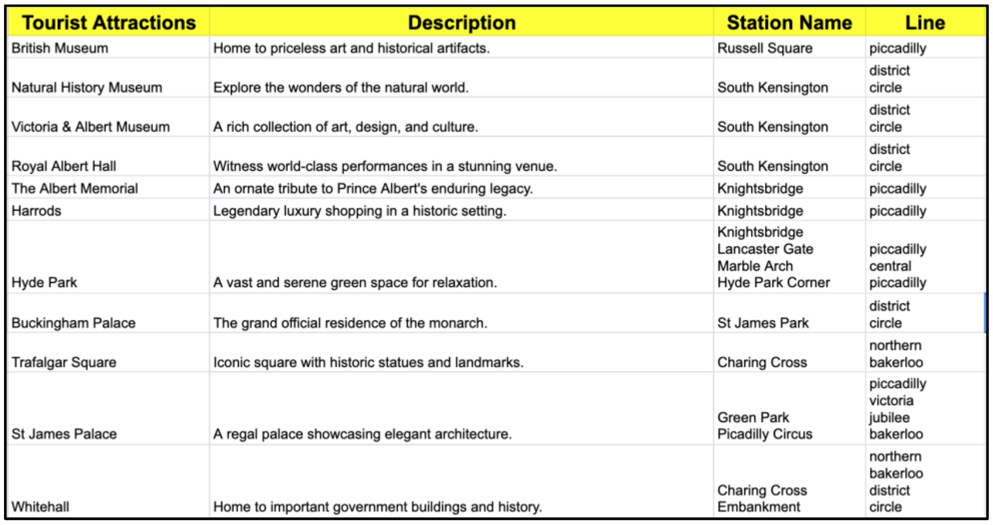
The station entity corresponds to information about the 121 stations of the London tube system in zone 1 and 2 that we are considering for our expert system development. The template for this entity has slots such as station name, line, zone, etc. In order to collect the data to populate the facts for this template, a list of the 121 stations under consideration was obtained from an online repository [1] by using python programming language to perform web scraping. Following this, the Transport for London Unified API or the TfL API [2] was used to collect additional information such as the lines that pass through a station, the neighbors of a tube station, etc. Figure 1 below illustrates a spreadsheet containing the data to populate facts for the station template -



Name	Line	Zone	neighbourStation	TransferStation
Aldgate	Metropolitan, Circle	1	Liverpool Street, Tower hill	No
Aldgate East	Hammersmith & City, District	1	Whitechapel, Liverpool Street, Tower hill	No
Angel	Northern	1	Old street, King's Cross & St Pancras International	No
Baker Street	Metropolitan, Bakerloo, Circle, Jubilee, Hammersmith & City	1	Regent's park, marylebone, Great Portland Street, Finchley road	Yes
Bank	Waterloo & City, Northern, Central	1	London bridge, moorgate, waterloo, St Paul's, liverpool street	Yes
Barbican	Metropolitan, Circle, Hammersmith & City	1	Farringdon, moorgate	Yes
Battersea Power Station	Northern	1	Nine Elms	Yes
Bayswater	District, Circle	1	Paddington, Notting hill gate	No
Blackfriars	District, Circle	1	Temple, Mansion House	No
Bond Street	Central, Jubilee	1	Oxford Circus, Mable Arch, Baker Street, Green Park	Yes
Borough	Northern	1	London bridge, Elephant & Castle	No
Cannon Street	District, Circle	1	Mansion House, Monument	Yes
Chancery Lane	Central	1	Holborn, St Paul's	No
Charing Cross	Bakerloo, Northern	1	Piccadilly Circus, Embankment, Leicester Square	Yes
Covent Garden	Piccadilly	1	Leicester Square, Holborn	No
Edgware Road (Bakerloo)	Bakerloo	1	Marylebone, Paddington	No
Edgware Road (Circle Line)	Hammersmith & City, District, Circle	1	Paddington, Baker Street	Yes
Embankment	District, Bakerloo, Northern, Circle	1	Temple, Waterloo, Westminster, Charging cross	Yes
Euston	Northern, Victoria	1	Camden Town, Mornington Crescent, Warren Street,King's Cross	Yes

Figure 14: Data to create facts for stations template

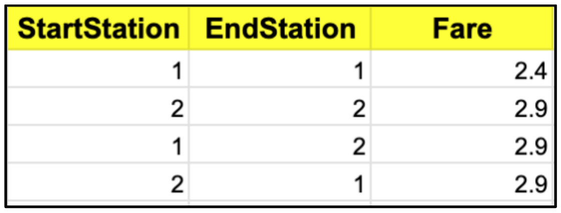
For the attractions entity, the template contains slots such as attraction name, description, etc. To create facts for the entity's template, a list of the 42 most relevant tourist attractions was obtained and then the Geocoding Google Maps API [3] was used to find the nearest relevant tube station for the attraction. The line information for the identified station was obtained by cross referencing already collected stations data as described at the beginning of this section. Python programming language in the Jupyter notebook environment was used to achieve this. Furthermore, to obtain the description for the attractions, the Google Places API [4] was used. Figure 2 below illustrates a spreadsheet containing the data collected using the procedures to populate the facts –



Tourist Attractions	Description	Station Name	Line
British Museum	Home to priceless art and historical artifacts.	Russell Square	piccadilly
Natural History Museum	Explore the wonders of the natural world.	South Kensington	district circle
Victoria & Albert Museum	A rich collection of art, design, and culture.	South Kensington	district circle
Royal Albert Hall	Witness world-class performances in a stunning venue.	South Kensington	district circle
The Albert Memorial	An ornate tribute to Prince Albert's enduring legacy.	Knightsbridge	piccadilly
Harrods	Legendary luxury shopping in a historic setting.	Knightsbridge	piccadilly
Hyde Park	A vast and serene green space for relaxation.	Knightsbridge Lancaster Gate Marble Arch Hyde Park Corner	piccadilly central piccadilly
Buckingham Palace	The grand official residence of the monarch.	St James Park	district circle
Trafalgar Square	Iconic square with historic statues and landmarks.	Charing Cross	northern bakerloo
St James Palace	A regal palace showcasing elegant architecture.	Green Park Picadilly Circus	piccadilly victoria jubilee bakerloo
Whitehall	Home to important government buildings and history.	Charing Cross Embankment	northern bakerloo district circle

Figure 15: Data to create facts for attractions template.

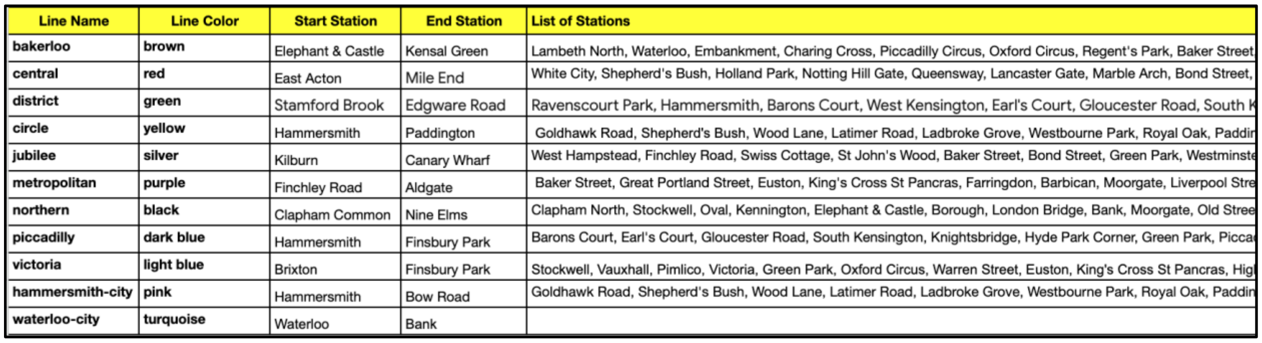
Thirdly, for the fare entity, the template will be populated with four facts only because we are considering only 2 zones for our expert system. The fares information included in the fact file is obtained from the fare table included in the project description document. Figure 3 below illustrates the spreadsheet containing fact data for fare template -



StartStation	EndStation	Fare
1	1	2.4
2	2	2.9
1	2	2.9
2	1	2.9

Figure 16: Data to create facts for attractions template.

For the final lines entity, the created template has information such as the line name, the start and ending stations, the list of all stations, etc. This information was collected using the Transport for London Unified API or the TfL API [2] and a snapshot of the collected facts is shown in figure 4 below -



Line Name	Line Color	Start Station	End Station	List of Stations
bakerloo	brown	Elephant & Castle	Kensal Green	Lambeth North, Waterloo, Embankment, Charing Cross, Piccadilly Circus, Oxford Circus, Regent's Park, Baker Street,
central	red	East Acton	Mile End	White City, Shepherd's Bush, Holland Park, Notting Hill Gate, Queensway, Lancaster Gate, Marble Arch, Bond Street,
district	green	Stamford Brook	Edgware Road	Ravenscourt Park, Hammersmith, Barons Court, West Kensington, Earl's Court, Gloucester Road, South K
circle	yellow	Hammersmith	Paddington	Goldhawk Road, Shepherd's Bush, Wood Lane, Latimer Road, Ladbroke Grove, Westbourne Park, Royal Oak, Paddin
jubilee	silver	Kilburn	Canary Wharf	West Hampstead, Finchley Road, Swiss Cottage, St John's Wood, Baker Street, Bond Street, Green Park, Westminste
metropolitan	purple	Finchley Road	Aldgate	Baker Street, Great Portland Street, Euston, King's Cross St Pancras, Farringdon, Barbican, Moorgate, Liverpool Stre
northern	black	Clapham Common	Nine Elms	Clapham North, Stockwell, Oval, Kennington, Elephant & Castle, Borough, London Bridge, Bank, Moorgate, Old Stree
piccadilly	dark blue	Hammersmith	Finsbury Park	Barons Court, Earl's Court, Gloucester Road, South Kensington, Knightsbridge, Hyde Park Corner, Green Park, Picca
victoria	light blue	Brixton	Finsbury Park	Stockwell, Vauxhall, Pimlico, Victoria, Green Park, Oxford Circus, Warren Street, Euston, King's Cross St Pancras, Hig
hammersmith-city	pink	Hammersmith	Bow Road	Goldhawk Road, Shepherd's Bush, Wood Lane, Latimer Road, Ladbroke Grove, Westbourne Park, Royal Oak, Paddin
waterloo-city	turquoise	Waterloo	Bank	

Figure 17: Data to create facts for lines template.

The entities described above were identified during phase one of project development. However, after detailed requirements analysis, some changes and modifications had to be made to the templates and entities. One of the major ones was the removal of the fares template and addition of the switch template described in the following paragraph. More details on changes made to facts and templates in the current phase 2 of project development are presented in section 4.2 below.

One of the key entities for our system is the switch template and the corresponding facts. The switch template, which is described in detail in the following sections, was obtained from the data in figure 9. The stations which fell under the category of transfer stations were extracted from this data along with the lines that could be switched at them. Following this, for each of the lines, the other lines that could be reached from them via switching stations were computed using a modified bfs (breadth-first search) algorithm. The pseudocode of this function is provided below -

```
function bfs_find_alternative_paths(graph, start, end, max_switches=4, max_alternatives=3):
  queue = create_empty_deque()
  visited = create_empty_set()
  alternatives = create_empty_list()

  # Enqueue the starting node with initial paths and switches
  queue.enqueue((start, [], [start], 0))

  # Explore the graph using BFS
  while queue is not empty:
      node, edge_path, node_path, switches = queue.dequeue()

      # Check if the destination node is reached
      if node == end:
          alternatives.append((edge_path, node_path))
```

```
            # Check if the maximum number of alternatives is reached
            if length(alternatives) >= max_alternatives:
                return alternatives

        # Mark the node as visited
        visited.add(node)

        # Explore neighbors
        for neighbor, edge_name in graph[node].items():
            if neighbor not in visited and switches < max_switches:
                new_edge_path = copy(edge_path)
                new_node_path = copy(node_path)
                new_switches = switches + 1

                # Update paths and enqueue the neighbor
                new_edge_path.append(edge_name)
                new_node_path.append(neighbor)
                queue.enqueue((neighbor, new_edge_path, new_node_path, new_switches))

    return alternatives
```

Instead of using bfs to calculate the route between stations on the fly, the possible switches between lines are anticipated, recorded, and fed into the knowledge base as facts using the above approach. This information was generated using the aforementioned data and python programming language and the entire code for this can be found in the iPython Notebook here [8]. The figure below illustrates a graph of switches and connections between the lines of the London Tube that was traversed to get the switch stations data.

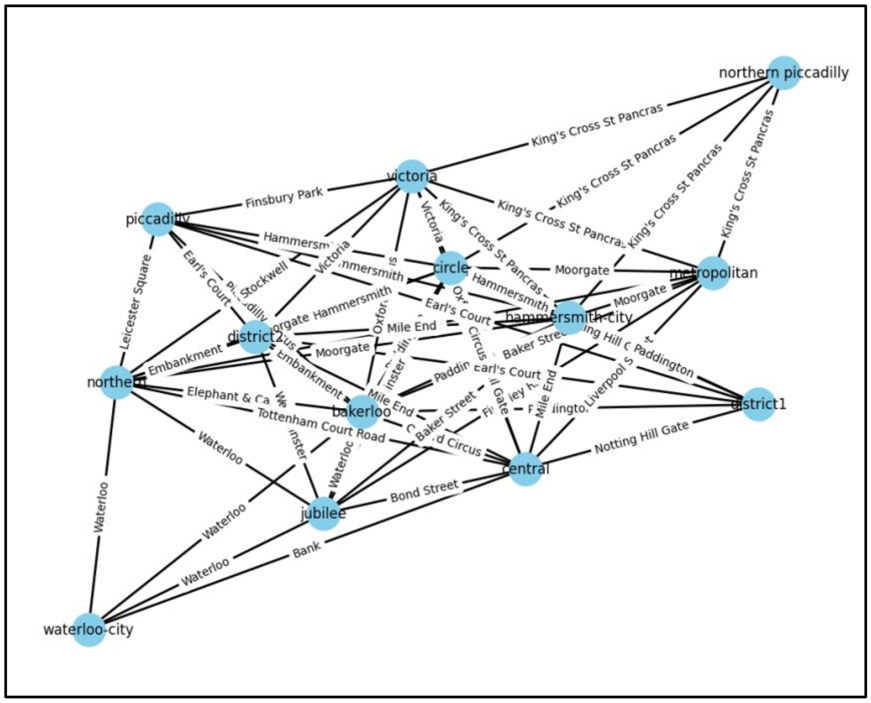


Figure 18: Data to create facts for lines template

Using the methods described in this section, suitable data was gathered and was structured into templates and facts to create the knowledge base of our expert system. The following sections provide more detail on this process as well as the modifications and changes made from phase one.

4.2. Revisions in Templates and Facts in Part-3

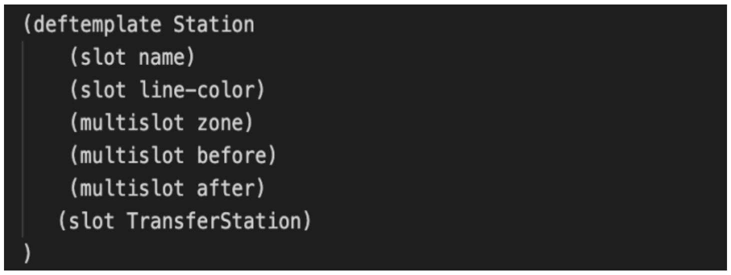
Templates are a fundamental concept in Clips and are used to define the structure of facts. They serve as a blueprint for creating instances of facts. Previously our system had the following templates: Station, Fare, Line, and Attractions. As we proceeded further with developing a rule-based system for getting routes and attraction information, the Station template was changed to store a list of inbound and outbound stations. Fare and Line facts were not needed so instead added a fact for Switch station and AttractionInfo which will be described further. Finally, the system has the following templates: Station, Switch, Attractions, and AttractionsInfo. These are described in the sections that follow.

For part 3, we have added LineDetails Facts and Templates to display the list of all stations and all transfer stations on a given line.

4.3. Templates in Clips

Below is the list of templates created:

- **Station:** This template is designed to capture various attributes of a station like:
    - Name: This slot is used to store the name of the station, which is of type STRING, allowing for text-based representation.
    - Line: This is a multislot, which means it can hold multiple values. It is used to store the names of the subway lines that enroute this station.
    - Zone: This is also a multislot and it stores integer values indicating the zone the station it lies in.
    - Stations Before: The "before" is a multislot for storing the list of all the stations before the current station
    - Stations After: The "after" slot is a multislot for storing the list of all the stations after the current station
    - TransferStation: This slot is also of type string and it stores yes or no depending on whether the station is transfer station or not.

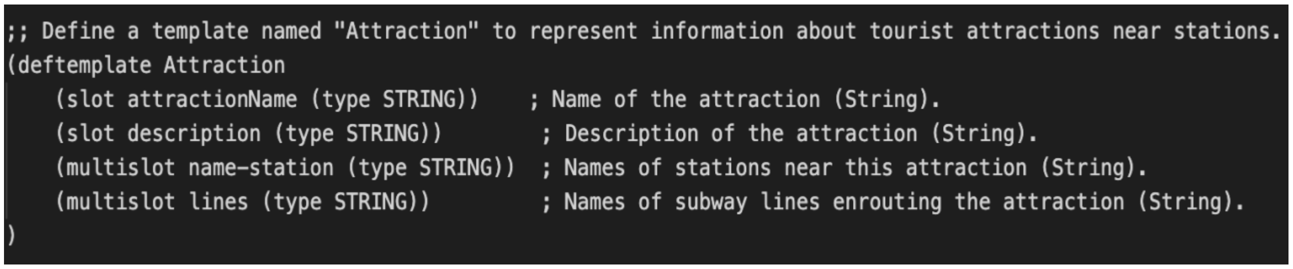


```
(deftemplate Station
    (slot name)
    (slot line-color)
    (multislot zone)
    (multislot before)
    (multislot after)
   (slot TransferStation)
)
```

Figure 19: Creating a template for Station entity

- **Attractions:** This template is designed to capture various attributes of line attractions:
    - Attraction name: This slot used to store the names of attractions in zone 1 and zone 2, which is of type STRING.
    - Description: This slot used to store the description of each attraction , which is of type STRING.
    - Station name: This multi slot used to store the names of stations nearest to this attraction, which is of type STRING.

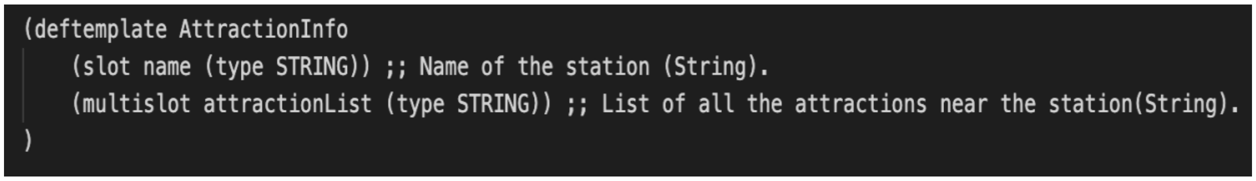
- ○ Line: This multi slot used to store the names of lines that are connected to the nearest station from this attraction, which is of type STRING.



```
;; Define a template named "Attraction" to represent information about tourist attractions near stations.
(deftemplate Attraction
    (slot attractionName (type STRING))    ; Name of the attraction (String).
    (slot description (type STRING))        ; Description of the attraction (String).
    (multislot name-station (type STRING))  ; Names of stations near this attraction (String).
    (multislot lines (type STRING))         ; Names of subway lines enrouting the attraction (String).
)
```

Figure 20: Creating a template for Attractions entity

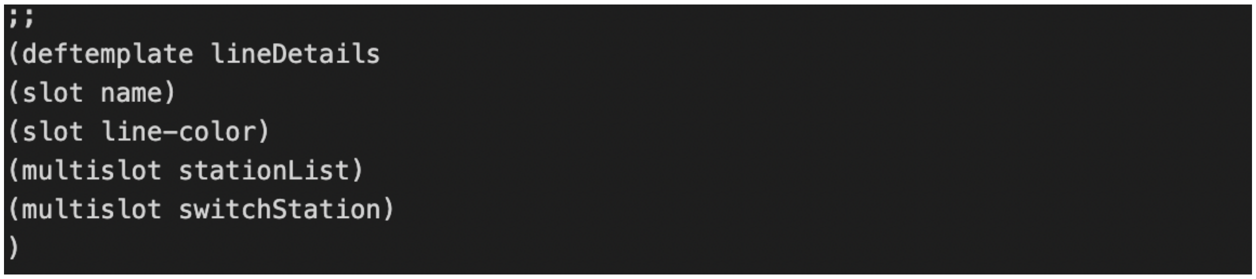
- **AttractionInfo:** This template is designed to capture list of attractions against each station:
    - ○ Name: This slot is used to store the name of the station, which is of type STRING, allowing for text-based representation.
    - ○ Attraction List: This multi slot used to store a list of attractions near the given station.



```
(deftemplate AttractionInfo
    (slot name (type STRING)) ;; Name of the station (String).
    (multislot attractionList (type STRING)) ;; List of all the attractions near the station(String).
)
```

Figure 21: Creating template for Attraction List against Stations

- **LineDetails:** This template is designed to capture list of all station on a particular line:
    - ○ Name: This slot is used to store the name of the line, which is of type STRING, allowing for text-based representation.
    - ○ Line-color: This slot is used to store the color of the line, which is of type STRING, allowing for text-based representation.
    - ○ Station List: This multi slot is used to store a list of all stations on the given line.
    - ○ Switch Station: This multi slot is used to store a list of all transfer stations on the given line.



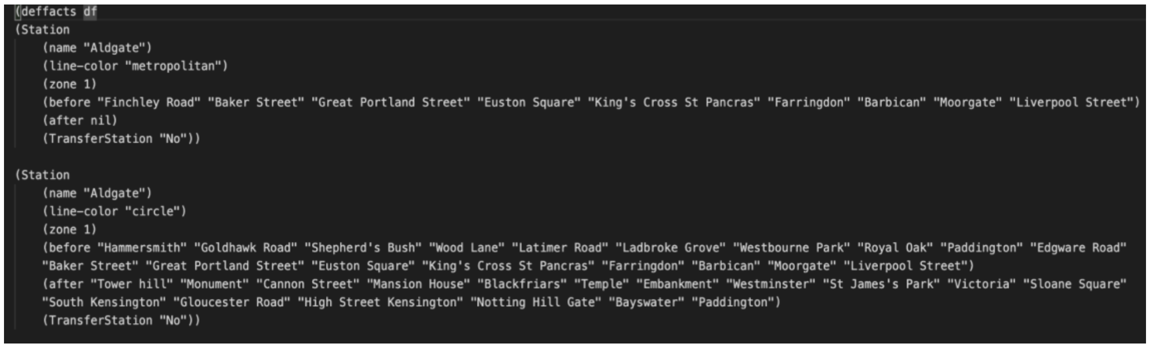
```
;;
(deftemplate lineDetails
(slot name)
(slot line-color)
(multislot stationList)
(multislot switchStation)
)
```

Figure 22: Creating template for Line Details

4.4. Facts in Clips

Facts are instances of templates. Facts are created by asserting them into Clips' working memory using the assert command, followed by the template name and values for each slot. They can also be defined using deffacts to define and initialize facts within the CLIPS working memory. They provide a convenient way to assert predefined sets of facts. Below are attached screenshots for instance of each facts created in our system:

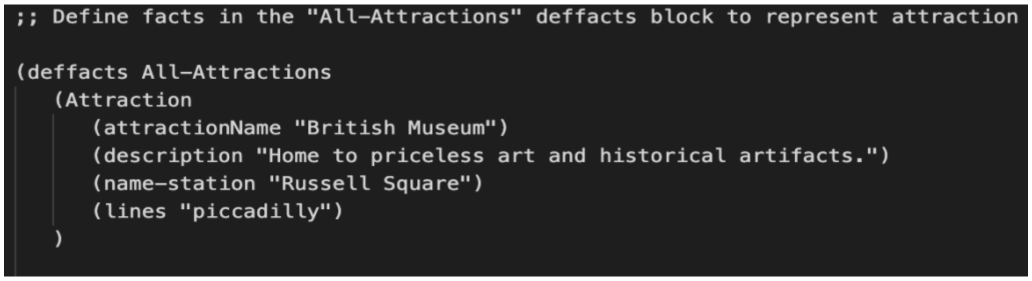
- **Station Facts:** They are structured in “station-facts” block to represent name of each station, lines that enroute each station, zone it lies in, the neighboring station this station and if its a transfer station or not.



```
(deffacts df
(Station
    (name "Aldgate")
    (line-color "metropolitan")
    (zone 1)
    (before "Finchley Road" "Baker Street" "Great Portland Street" "Euston Square" "King's Cross St Pancras" "Farringdon" "Barbican" "Moorgate" "Liverpool Street")
    (after nil)
    (TransferStation "No"))

(Station
    (name "Aldgate")
    (line-color "circle")
    (zone 1)
    (before "Hammersmith" "Goldhawk Road" "Shepherd's Bush" "Wood Lane" "Latimer Road" "Ladbroke Grove" "Westbourne Park" "Royal Oak" "Paddington" "Edgware Road"
    "Baker Street" "Great Portland Street" "Euston Square" "King's Cross St Pancras" "Farringdon" "Barbican" "Moorgate" "Liverpool Street")
    (after "Tower hill" "Monument" "Cannon Street" "Mansion House" "Blackfriars" "Temple" "Embankment" "Westminster" "St James's Park" "Victoria" "Sloane Square"
    "South Kensington" "Gloucester Road" "High Street Kensington" "Notting Hill Gate" "Bayswater" "Paddington")
    (TransferStation "No"))
```

Figure 23: Adding facts for Station template

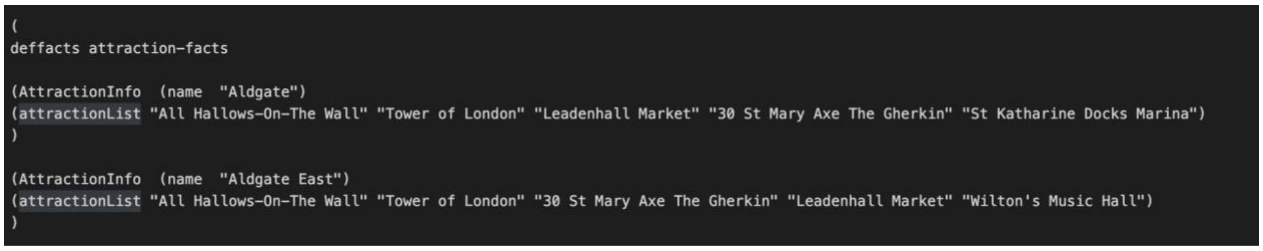
- **Attractions Facts:** They are structured in "All-Attractions" deffacts block to represent attraction name, its description, the nearest station to that attraction and lines that enroute the nearest station, for each attraction in zone 1 & zone 2 based on the template defined.



```
;; Define facts in the "All-Attractions" deffacts block to represent attraction

(deffacts All-Attractions
   (Attraction
      (attractionName "British Museum")
      (description "Home to priceless art and historical artifacts.")
      (name-station "Russell Square")
      (lines "piccadilly")
   )
```

Figure 24: Adding facts for Attraction template

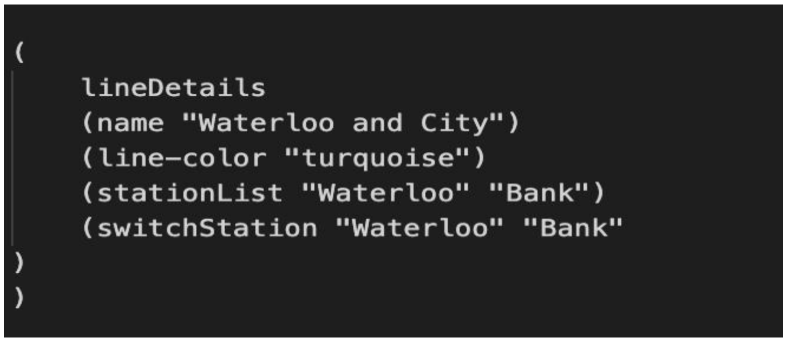
- **AttractionInfo Facts:** They are structured in "attraction-facts" deffacts block to represent station name and list of all attractions near it, for each station in zone 1 & zone 2 based on the template defined.



```
(
deffacts attraction-facts

(AttractionInfo  (name  "Aldgate")
(attractionList "All Hallows-On-The Wall" "Tower of London" "Leadenhall Market" "30 St Mary Axe The Gherkin" "St Katharine Docks Marina")
)

(AttractionInfo  (name  "Aldgate East")
(attractionList "All Hallows-On-The Wall" "Tower of London" "30 St Mary Axe The Gherkin" "Leadenhall Market" "Wilton's Music Hall")
)
```

Figure 25: Adding facts for Attraction template

- **LineDetails Facts:** They are structured in "lineDetails-facts" deffacts block to represent line name, line color, list of all stations on that particular line and list of all switch stations on it, for each line in zone 1 & zone 2 based on the template defined.



```
(
    lineDetails
    (name "Waterloo and City")
    (line-color "turquoise")
    (stationList "Waterloo" "Bank")
    (switchStation "Waterloo" "Bank"
)
)
```

Figure 26: Adding facts for Line Details template

Below is the diagram to demonstrate the relation between facts and templates -

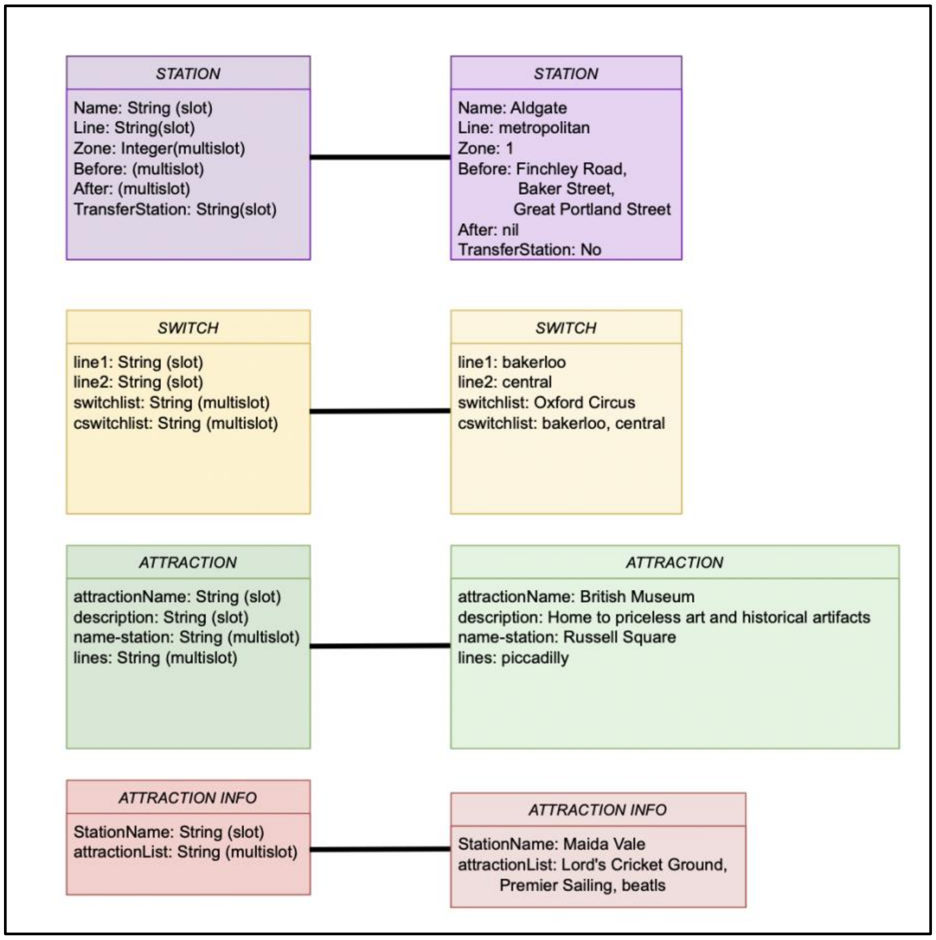
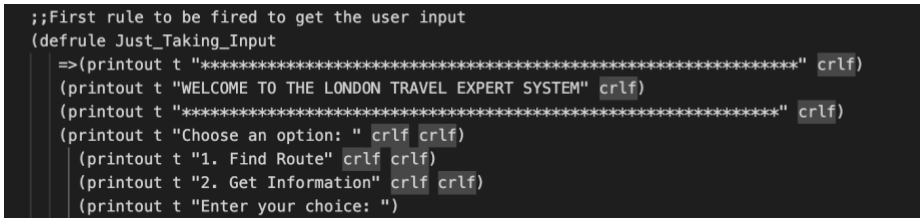

STATION
Name: String (slot)
Line: String(slot)
Zone: Integer(multislot)
Before: (multislot)
After: (multislot)
TransferStation: String(slot)
STATION
Name: Aldgate
Line: metropolitan
Zone: 1
Before: Finchley Road, Baker Street, Great Portland Street
After: nil
TransferStation: No
SWITCH
line1: String (slot)
line2: String (slot)
switchlist: String (multislot)
cswitchlist: String (multislot)
SWITCH
line1: bakerloo
line2: central
switchlist: Oxford Circus
cswitchlist: bakerloo, central
ATTRACTION
attractionName: String (slot)
description: String (slot)
name-station: String (multislot)
lines: String (multislot)
ATTRACTION
attractionName: British Museum
description: Home to priceless art and historical artifacts
name-station: Russell Square
lines: piccadilly
ATTRACTION INFO
StationName: String (slot)
attractionList: String (multislot)
ATTRACTION INFO
StationName: Maida Vale
attractionList: Lord's Cricket Ground, Premier Sailing, beatls

Figure 27: Diagrammatic Illustration of Templates and Example Facts

5. Design and Implementation of CLIPS Rules

CLIPS is a system development tool used to create rule-based systems. Rules are structured with condition and the action part. The action part contains the tasks to be executed when the condition is satisfied. The condition part of a rule uses pattern matching to identify specific patterns in the data or facts already stored within the system. In our system following rules are defined to get routes and information on attractions in London. The first rule fired is for taking user input based on the choice given.

- **Input rule:** It prompts the user with a multiple choice on whether he/she wants information on station or route between start and destination station. According to the user entered choice it fires another. Below is the screenshot for the same.

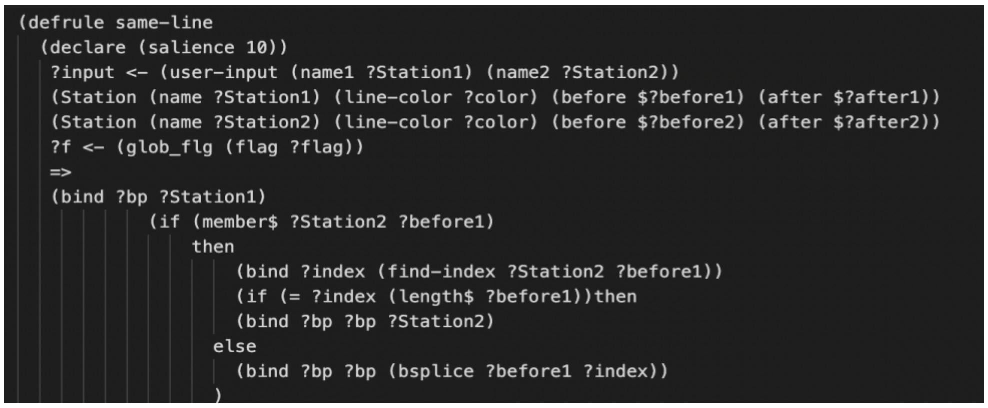


```
;;First rule to be fired to get the user input
(defrule Just_Taking_Input
  =>(printout t "***************************************************************" crlf)
  (printout t "WELCOME TO THE LONDON TRAVEL EXPERT SYSTEM" crlf)
  (printout t "***************************************************************" crlf)
  (printout t "Choose an option: " crlf crlf)
    (printout t "1. Find Route" crlf crlf)
    (printout t "2. Get Information" crlf crlf)
    (printout t "Enter your choice: ")
```

Figure 28: User Input Rule

Next, if the user selects choice one the route rule is fired -

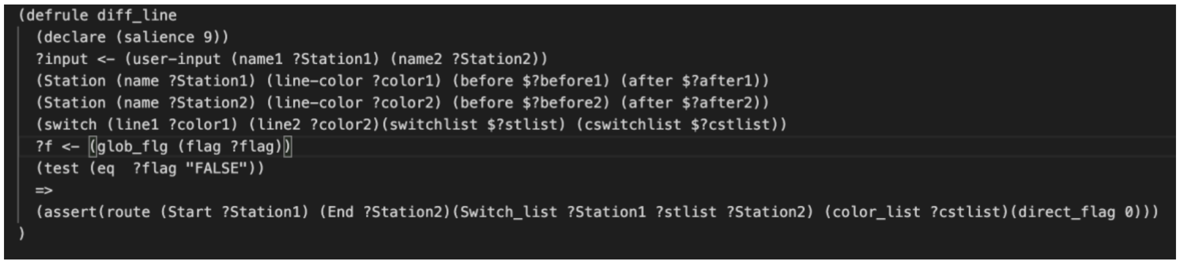
- **Routes:** This rule takes start station and end station as input and calculates all the possible routes between them. First it checks in the station facts if both the entered stations are on the same line and returns the list of stations in order from start to end. The screenshot of that rule is as below:



```
(defrule same-line
  (declare (salience 10))
   ?input <- (user-input (name1 ?Station1) (name2 ?Station2))
   (Station (name ?Station1) (line-color ?color) (before $?before1) (after $?after1))
   (Station (name ?Station2) (line-color ?color) (before $?before2) (after $?after2))
   ?f <- (glob_flg (flag ?flag))
   =>
   (bind ?bp ?Station1)
            (if (member$ ?Station2 ?before1)
                then
                    (bind ?index (find-index ?Station2 ?before1))
                    (if (= ?index (length$ ?before1))then
                    (bind ?bp ?bp ?Station2)
                  else
                    (bind ?bp ?bp (bsplice ?before1 ?index))
                  )
```

Figure 29: Rule for stations on same line

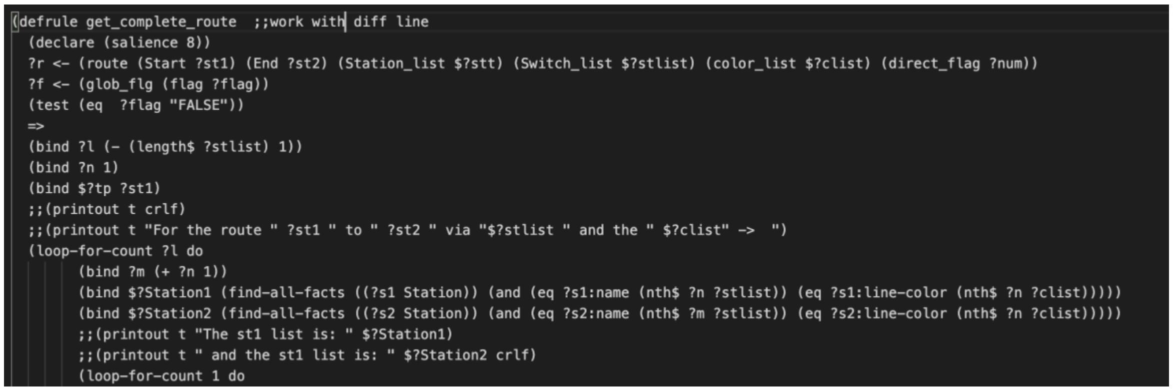
Further, if they are not on the same line it fires the diff_line rule to assert new facts containing all possible routes based on the stations and switch facts. The code screen shot for same is as below:



```
(defrule diff_line
  (declare (salience 9))
  ?input <- (user-input (name1 ?Station1) (name2 ?Station2))
  (Station (name ?Station1) (line-color ?color1) (before $?before1) (after $?after1))
  (Station (name ?Station2) (line-color ?color2) (before $?before2) (after $?after2))
  (switch (line1 ?color1) (line2 ?color2)(switchlist $?stlist) (cswitchlist $?cstlist))
  ?f <- (glob_flg (flag ?flag))
  (test (eq  ?flag "FALSE"))
  =>
  (assert(route (Start ?Station1) (End ?Station2)(Switch_list ?Station1 ?stlist ?Station2) (color_list ?cstlist)(direct_flag 0)))
)
```

Figure 30: Rule for stations on different lines

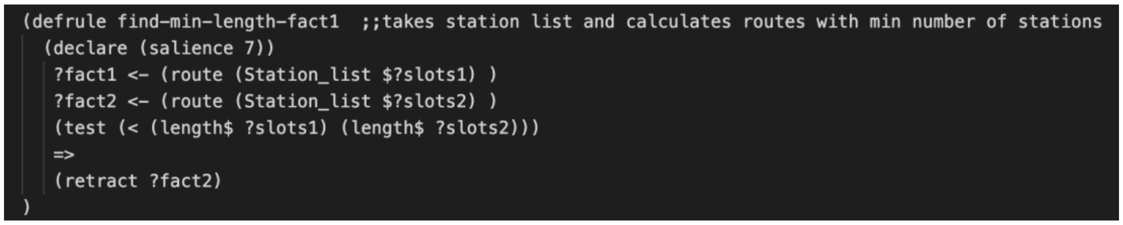
Now once these facts are asserted into the system get_complete_route will calculate the optimal route and return it to the user. In order to calculate the optimal route it will retract all the facts that have a minimum number of stations and eliminate the rest. Now if there is more than one route with minimum stations; in that case it will check for the number of line switches and select the one with minimum switches. And in the end if there are again more than one possible route it will prompt all the possible options. Instead of going through standard BFS we choose this method for optimization to handle the features of phase 3. Below is the screenshot of the same:



```
(defrule get_complete_route  ;;work with diff line
  (declare (salience 8))
  ?r <- (route (Start ?st1) (End ?st2) (Station_list $?stt) (Switch_list $?stlist) (color_list $?clist) (direct_flag ?num))
  ?f <- (glob_flg (flag ?flag))
  (test (eq  ?flag "FALSE"))
  =>
  (bind ?l (- (length$ ?stlist) 1))
  (bind ?n 1)
  (bind $?tp ?st1)
  ;;(printout t crlf)
  ;;(printout t "For the route " ?st1 " to " ?st2 " via "$?stlist " and the " $?clist" ->  ")
  (loop-for-count ?l do
        (bind ?m (+ ?n 1))
        (bind $?Station1 (find-all-facts ((?s1 Station)) (and (eq ?s1:name (nth$ ?n ?stlist)) (eq ?s1:line-color (nth$ ?n ?clist)))))
        (bind $?Station2 (find-all-facts ((?s2 Station)) (and (eq ?s2:name (nth$ ?m ?stlist)) (eq ?s2:line-color (nth$ ?n ?clist)))))
        ;;(printout t "The st1 list is: " $?Station1)
        ;;(printout t " and the st1 list is: " $?Station2 crlf)
        (loop-for-count 1 do
```

Figure 31: Main Rule to Calculate Routes

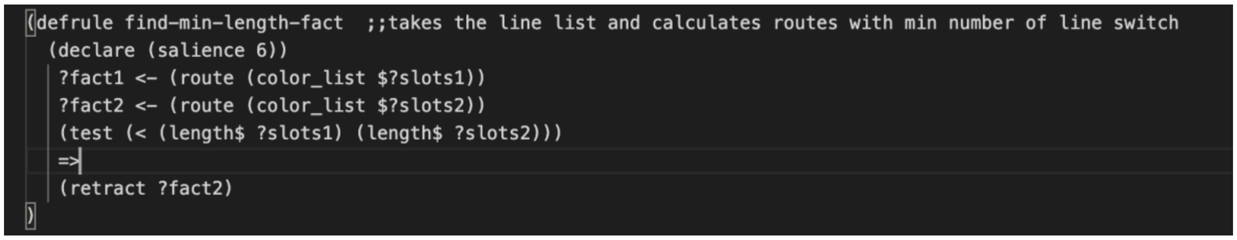
In addition to these, find-min-length-fact1 will take a station list and calculate the routes with the minimum number of stations.



```
(defrule find-min-length-fact1  ;;takes station list and calculates routes with min number of stations
  (declare (salience 7))
   ?fact1 <- (route (Station_list $?slots1) )
   ?fact2 <- (route (Station_list $?slots2) )
   (test (< (length$ ?slots1) (length$ ?slots2)))
   =>
   (retract ?fact2)
 )
```

Figure 32: Rule to calculate min stations

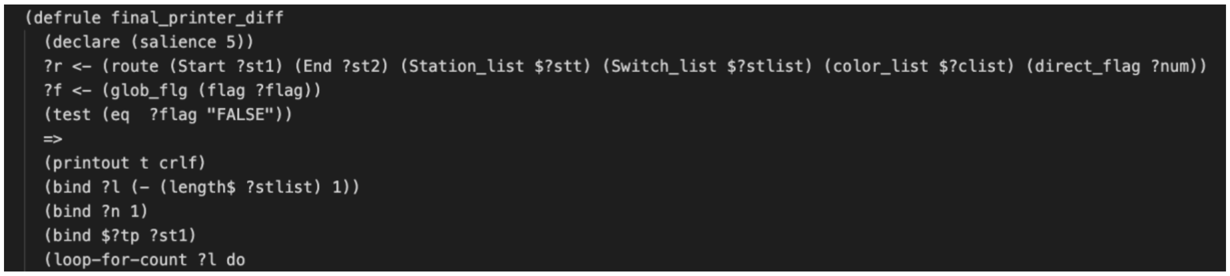
The find-min-length-fact rule is used to take the line list and calculate routes with a minimum number of line switches.



```
(defrule find-min-length-fact  ;;takes the line list and calculates routes with min number of line switch
  (declare (salience 6))
   ?fact1 <- (route (color_list $?slots1))
   ?fact2 <- (route (color_list $?slots2))
   (test (< (length$ ?slots1) (length$ ?slots2)))
   =>
   (retract ?fact2)
)
```

Figure 33: Rule to calculate min line switch

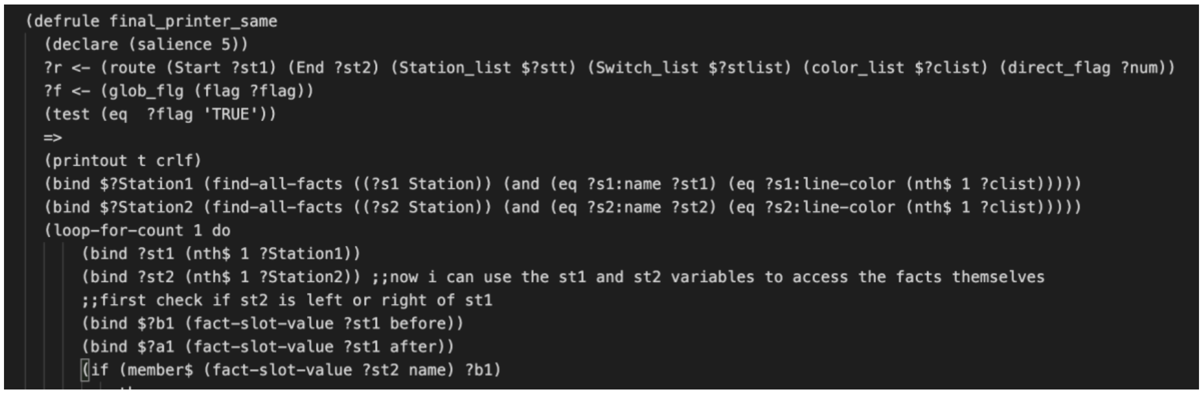
Now, to print the output of the route final_printer_diff will print the output for the case when start and end stations are on different lines.



```
(defrule final_printer_diff
  (declare (salience 5))
  ?r <- (route (Start ?st1) (End ?st2) (Station_list $?stt) (Switch_list $?stlist) (color_list $?clist) (direct_flag ?num))
  ?f <- (glob_flg (flag ?flag))
  (test (eq  ?flag "FALSE"))
  =>
  (printout t crlf)
  (bind ?l (- (length$ ?stlist) 1))
  (bind ?n 1)
  (bind $?tp ?st1)
  (loop-for-count ?l do
```

Figure 34: Rule to print output when different lines

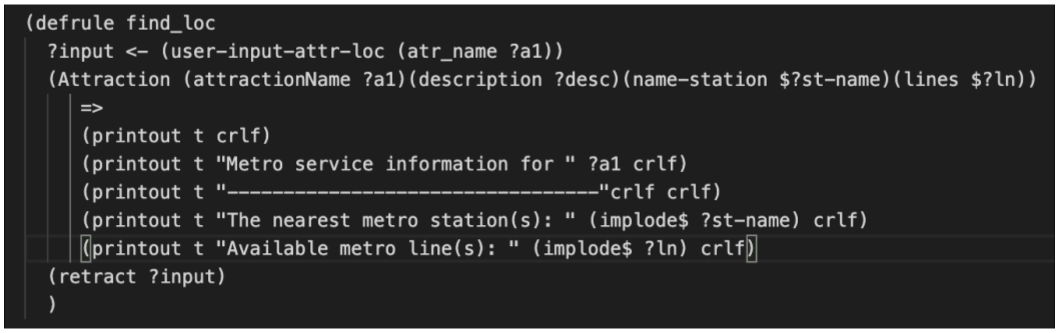
Now, to print the output of the route final_printer_same will print the output for the case when start and end stations are on the same line.



```
(defrule final_printer_same
  (declare (salience 5))
  ?r <- (route (Start ?st1) (End ?st2) (Station_list $?stt) (Switch_list $?stlist) (color_list $?clist) (direct_flag ?num))
  ?f <- (glob_flg (flag ?flag))
  (test (eq  ?flag 'TRUE'))
  =>
  (printout t crlf)
  (bind $?Station1 (find-all-facts ((?s1 Station)) (and (eq ?s1:name ?st1) (eq ?s1:line-color (nth$ 1 ?clist)))))
  (bind $?Station2 (find-all-facts ((?s2 Station)) (and (eq ?s2:name ?st2) (eq ?s2:line-color (nth$ 1 ?clist)))))
  (loop-for-count 1 do
      (bind ?st1 (nth$ 1 ?Station1))
      (bind ?st2 (nth$ 1 ?Station2)) ;;now i can use the st1 and st2 variables to access the facts themselves
      ;;first check if st2 is left or right of st1
      (bind $?b1 (fact-slot-value ?st1 before))
      (bind $?a1 (fact-slot-value ?st1 after))
      (if (member$ (fact-slot-value ?st2 name) ?b1)
```

Figure 35: Rule to print output when same line

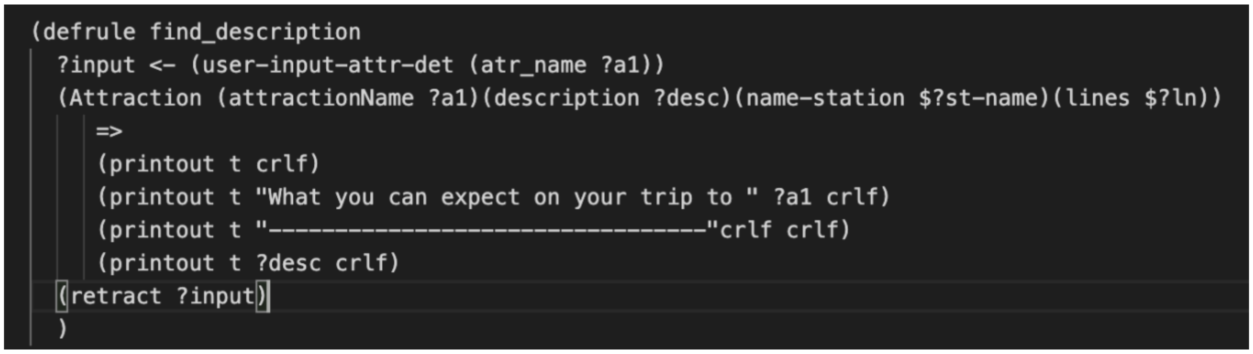
- **Attractions:** This has three more rules based on the option selected. When the user selects the get information option it will again prompt 3 options as: Find Nearest Stations to Attraction, Find Information on Attractions, and Find Attractions near a Station. If user selects option 1 it will fire find_loc rule and based on the Attraction fact fetch details on nearest metro station to the attraction entered by the user.



```
(defrule find_loc
  ?input <- (user-input-attr-loc (atr_name ?a1))
  (Attraction (attractionName ?a1)(description ?desc)(name-station $?st-name)(lines $?ln))
     =>
     (printout t crlf)
     (printout t "Metro service information for " ?a1 crlf)
     (printout t "--------------------------------"crlf crlf)
     (printout t "The nearest metro station(s): " (implode$ ?st-name) crlf)
     (printout t "Available metro line(s): " (implode$ ?ln) crlf)
  (retract ?input)
  )
```

Figure 36: Rule to find metro station closest to the attraction

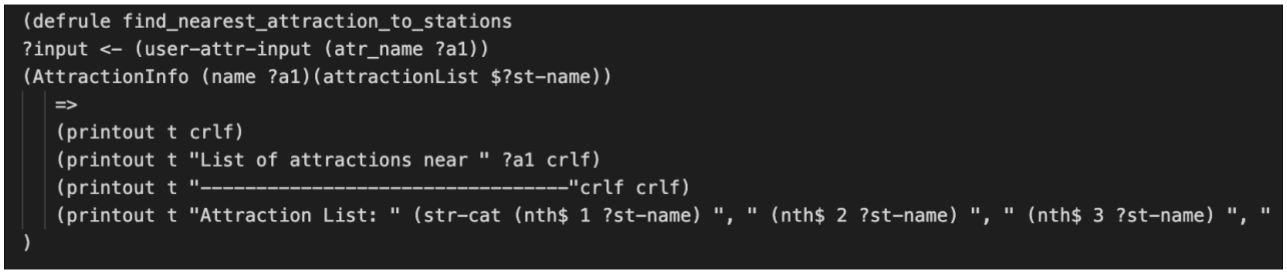
For option 2 it will fire the find_description rule and based on Attraction fact it will fetch the description of the attraction entered by the user.



```
(defrule find_description
  ?input <- (user-input-attr-det (atr_name ?a1))
  (Attraction (attractionName ?a1)(description ?desc)(name-station $?st-name)(lines $?ln))
     =>
     (printout t crlf)
     (printout t "What you can expect on your trip to " ?a1 crlf)
     (printout t "--------------------------------"crlf crlf)
     (printout t ?desc crlf)
  (retract ?input)
  )
```

Figure 37: Rule to find attraction details

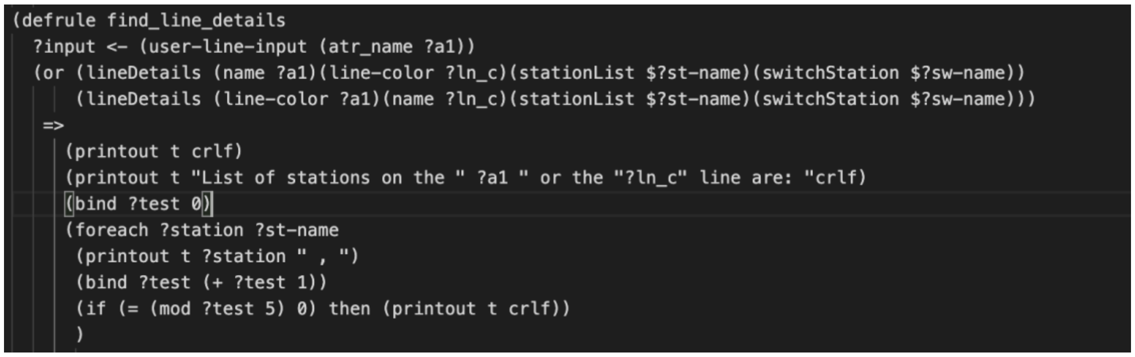
For option 3 it will fire the find_nearest_attraction_to_stations rule and based on the AttractionInfo fact it will fetch the list of top 5 attractions near the entered metro station.



```
(defrule find_nearest_attraction_to_stations
?input <- (user-attr-input (atr_name ?a1))
(AttractionInfo (name ?a1)(attractionList $?st-name))
   =>
   (printout t crlf)
   (printout t "List of attractions near " ?a1 crlf)
   (printout t "--------------------------------"crlf crlf)
   (printout t "Attraction List: " (str-cat (nth$ 1 ?st-name) ", " (nth$ 2 ?st-name) ", " (nth$ 3 ?st-name) ", "
)
```

Figure 38: Rule to find closest attractions near metro station

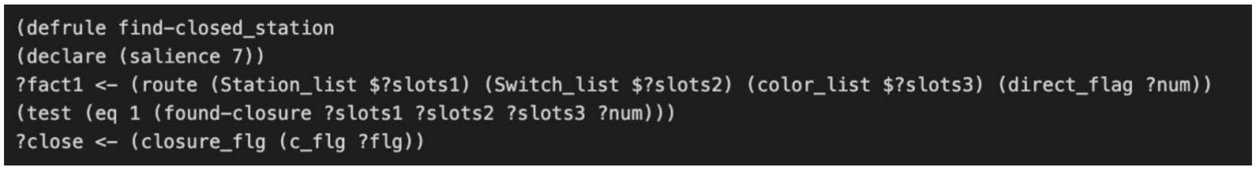
- **Line Details:** This has one rule based on the option selected. When the user selects the get information option and then selects option 5 (get line details) from the sub-menu fire find_line_details rule and based on the lineDetails fact fetch details on all the stations and transfer stations on the given line.



```
(defrule find_line_details
  ?input <- (user-line-input (atr_name ?a1))
  (or (lineDetails (name ?a1)(line-color ?ln_c)(stationList $?st-name)(switchStation $?sw-name))
      (lineDetails (line-color ?a1)(name ?ln_c)(stationList $?st-name)(switchStation $?sw-name)))
   =>
    (printout t crlf)
    (printout t "List of stations on the " ?a1 " or the "?ln_c" line are: "crlf)
    (bind ?test 0)
    (foreach ?station ?st-name
     (printout t ?station " , ")
     (bind ?test (+ ?test 1))
     (if (= (mod ?test 5) 0) then (printout t crlf))
     )
```

Figure 39: Rule to find closest attractions near metro station

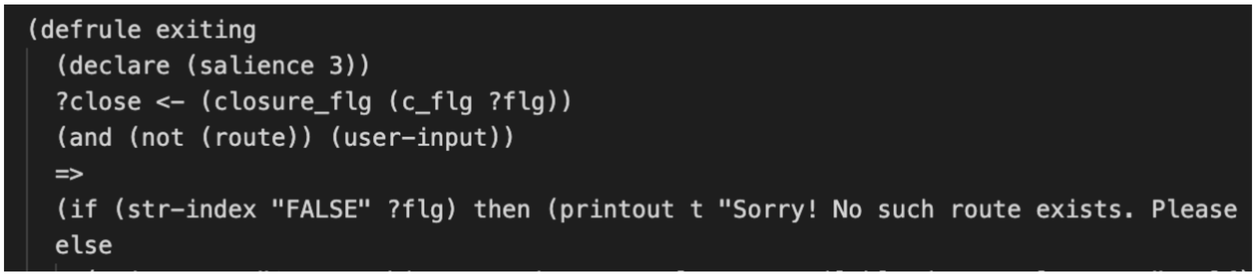
- **Closed Stations:** This rule has been created to identify all those routes that contain closure stations and revoke then so that the final output to the user is not a route that includes any closed stations. This rule takes as input, the asserted routes and invokes the function found-closure which checks for the existence of out of service stations and if such a route is identified, it is revoked. The rule is shown in the s screenshot below -



```
(defrule find-closed_station
(declare (salience 7))
?fact1 <- (route (Station_list $?slots1) (Switch_list $?slots2) (color_list $?slots3) (direct_flag ?num))
(test (eq 1 (found-closure ?slots1 ?slots2 ?slots3 ?num)))
?close <- (closure_flg (c_flg ?flg))
```

Figure 40: Rule to find alternate routes in case of closed stations

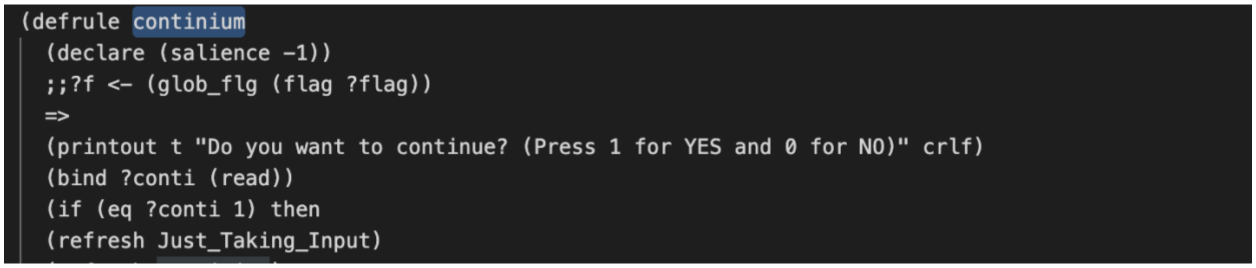
- **Error in Input:** This rule handles the case where either, the name entered by the user corresponds to no station in the facts of the exert system or no available route between the entered stations exist. A screenshot depicting the rule is shown below -



```
(defrule exiting
  (declare (salience 3))
  ?close <- (closure_flg (c_flg ?flg))
  (and (not (route)) (user-input))
  =>
  (if (str-index "FALSE" ?flg) then (printout t "Sorry! No such route exists. Please
  else
```

Figure 41: Rule to print put error message

- **Looping to Main Menu:** This rule ensures that after every query of the user is done executing, the system loops back to the main menu instead of exiting. Usre input is asked and if the user chooses to continue, the program loops back to the main menu, else the program is exited.



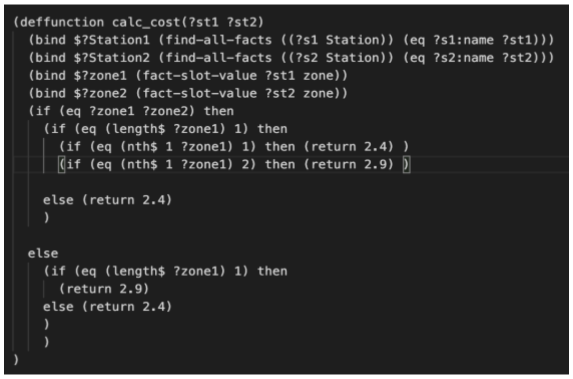
```
(defrule continium
  (declare (salience -1))
  ;;?f <- (glob_flg (flag ?flag))
  =>
  (printout t "Do you want to continue? (Press 1 for YES and 0 for NO)" crlf)
  (bind ?conti (read))
  (if (eq ?conti 1) then
  (refresh Just_Taking_Input)
```

Figure 42: Rule to loop back to main menu given user input

- **Miscelaneous Rules:** In addition to the rules mentioned above, there are some additional ones that aid in system utility. All of these rules handle the cases where the user input provided does not correspond to a station, attraction, or line. Instead of the system simply exiting execution, these rules ensure that a relevant error message is printed out and the user is asked if they would like to loop back to the main menu.

6. Supporting CLIPS Functions

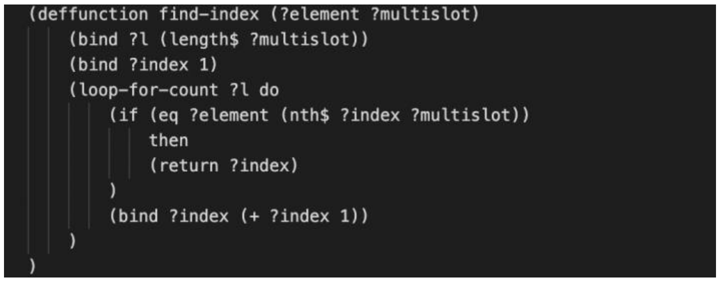
- **Functions:** Apart from rules some functions have been implemented. They are as follows:
  - **Calc_cost:** This function calculates the cost of travel from one station to another in route based on zone positioning.



```
(deffunction calc_cost(?st1 ?st2)
  (bind $?Station1 (find-all-facts ((?s1 Station)) (eq ?s1:name ?st1)))
  (bind $?Station2 (find-all-facts ((?s2 Station)) (eq ?s2:name ?st2)))
  (bind $?zone1 (fact-slot-value ?st1 zone))
  (bind $?zone2 (fact-slot-value ?st2 zone))
  (if (eq ?zone1 ?zone2) then
    (if (eq (length$ ?zone1) 1) then
      (if (eq (nth$ 1 ?zone1) 1) then (return 2.4) )
      (if (eq (nth$ 1 ?zone1) 2) then (return 2.9) )

    else (return 2.4)
    )

  else
    (if (eq (length$ ?zone1) 1) then
      (return 2.9)
    else (return 2.4)
    )
    )
)
```

Figure 43: Calculate cost function

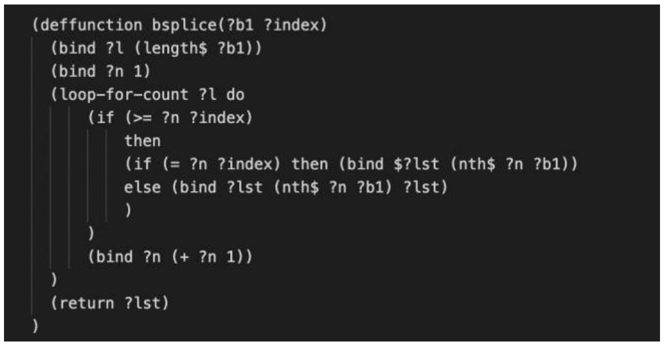
  - **Find_index:** Calculates the position of station in the before and after list of another station



```
(deffunction find-index (?element ?multislot)
    (bind ?l (length$ ?multislot))
    (bind ?index 1)
    (loop-for-count ?l do
        (if (eq ?element (nth$ ?index ?multislot))
            then
            (return ?index)
        )
        (bind ?index (+ ?index 1))
    )
)
```

Figure 44: Find Index Function

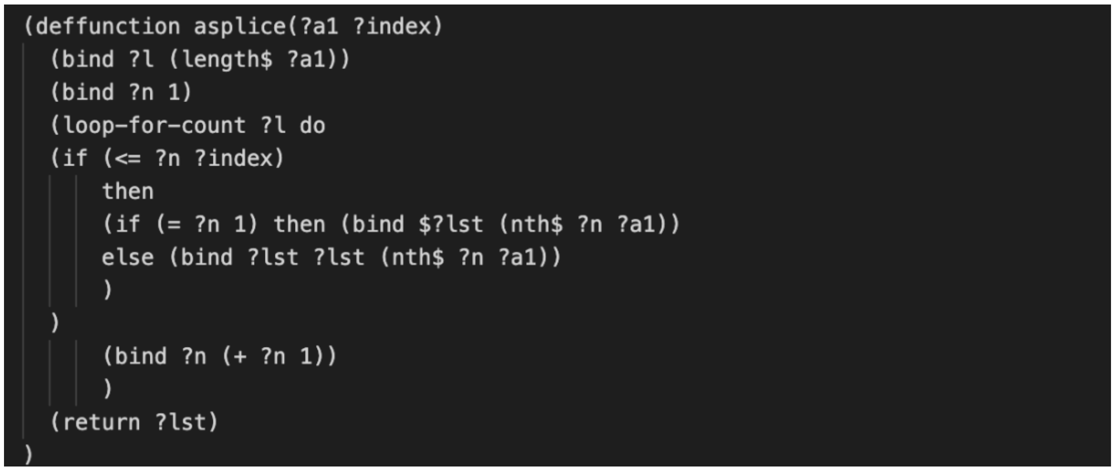
  - **Bsplice:** Calculates the subset of stations from the before travel list that needs to be traveled to get to the destination station.



```
(deffunction bsplice(?b1 ?index)
  (bind ?l (length$ ?b1))
  (bind ?n 1)
  (loop-for-count ?l do
      (if (>= ?n ?index)
          then
          (if (= ?n ?index) then (bind $?lst (nth$ ?n ?b1))
          else (bind ?lst (nth$ ?n ?b1) ?lst)
          )
      )
      (bind ?n (+ ?n 1))
  )
  (return ?lst)
)
```

Figure 45: Function to calculate the subset of stations from before travel list

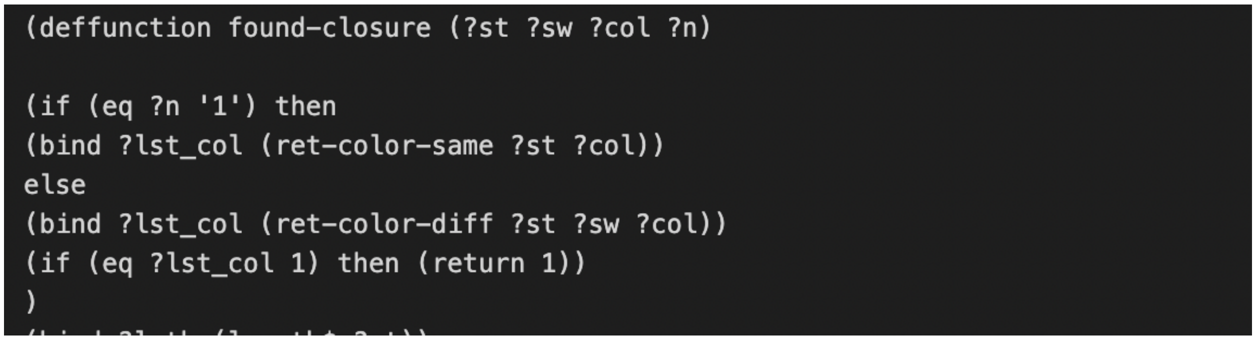
  - **Asplice:** Calculates the subset of stations from the after travel list that needs to be traveled to get to the destination station.



```
(deffunction asplice(?a1 ?index)
  (bind ?l (length$ ?a1))
  (bind ?n 1)
  (loop-for-count ?l do
  (if (<= ?n ?index)
      then
      (if (= ?n 1) then (bind $?lst (nth$ ?n ?a1))
      else (bind ?lst ?lst (nth$ ?n ?a1))
      )
  )
      (bind ?n (+ ?n 1))
      )
  (return ?lst)
)
```

Figure 46: Function to calculate the subset of stations from after travel list

- **Found-closure:** This function is designed to, given a route that has been found between user-input stations, check if that route has any of the specified station closures. If closures exist, a flag variable is returned back to the calling rule which then revokes this rule.



```
(deffunction found-closure (?st ?sw ?col ?n)

(if (eq ?n '1') then
(bind ?lst_col (ret-color-same ?st ?col))
else
(bind ?lst_col (ret-color-diff ?st ?sw ?col))
(if (eq ?lst_col 1) then (return 1))
)
```

Figure 47: Function to find the alternate route incase of closed station

7. Conclusions

In part-3 of the system development, we developed an interactive system which will give information on routes and attractions in London while also bypassing and taking care of station closures. After defining templates and populating facts based on information collected from diverse sources, we designed and implemented rules for developing rule-based systems and knowledge-based applications.

In conclusion, our use of CLIPS for the London Tube expert system successfully integrates advanced rule-based systems and knowledge-based applications. Through systematic entity identification and rule development, we've created a robust and practical platform. This application highlights the efficiency of CLIPS in managing complex rule structures, demonstrating its effectiveness in providing straightforward navigation and information retrieval within the London Tube network.

8. References

[1] Wikipedia contributors. (2023, September). List of London Underground stations. Wikipedia. URL: https://en.wikipedia.org/wiki/List_of_London_Underground_stations

[2] Transport for London. (2023, September). TfL API. URL: https://api.tfl.gov.uk/

[3] Google. (2023, September). Google Maps Geocoding API. URL: https://developers.google.com/maps/documentation/geocoding/start

[4] Google. (2023, September). Google Maps Places API. URL: https://developers.google.com/maps/documentation/places/web-service/overview

[5] *CLIPS User Guide Chapter 2*. (n.d.). https://redirect.cs.umbc.edu/portal/clips/usersguide/ug2.html

[6] GeeksforGeeks. (2023, June 9). *Breadth first search or BFS for a graph*. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[7] Robinson, P. (n.d.). *CLIPS Tutorial 1*. https://kcir.pwr.edu.pl/~witold/ai/CLIPS_tutorial/CLIPS_tutorial_1.html

[8] Chadha, H. (2023, November 7). *BFS for London Tube Navigation System*. Google Colaboratory. https://colab.research.google.com/drive/1a3qAhu9FnyhL9GJln70Z37V9KBcY0Cxz#scrollTo=-pP8LE1BWJT5