**D - Main usage:**

To begin, here is the main usage of the program:

Firstly, if I want for instance to go from Pine Street to the Airport using the shortest route, I can type that command.

It will output the shortest route using the stops names.

Secondly, I can ask the program how long it will take. For instance, it takes 24 minutes to go from King Street to Logan Road.

**A - Declaring the stops:**

The first step of the code is the most obvious and the easiest: declaring the stops.

Here we declare every stop, along with a list of the lines passing through it.

(Shows to or three examples on the map)

**D - Declaring adjacent stops:**

Then we need to tell the program which of these stops are adjacent to eachother.

Firstly, we created a subrule to avoid having to declare them both ways.

Then we just declared the stops, here we can see the stops of the underground 1.

(Shows some examples on the map)

**A - The sameLine rule:**

The sameLine is one of the extra rules we were able to create to add some functionality to the program.

With this rule, we can check if two stops share the same line.

This rule is really to understand, if Line1 and Line2 are both members of Line, it means both stops are on the same line.

(Showing the stops on the map)

For example, if I check if Saffron Road and Sutton Road share a line, it will output u1.

But if I check for the Airport and Pine Street, it will output false.

**D - The findAllStops rule:**

This rule allows us to input a line, and to get as an output the list of every stop on this line.

For instance, if I try with the line bus 2, which is the shortest one, it will give the list of the 4 stops.

**A - The numberOfLines rule:**

This rule is basically the opposite of the last one, it takes a stop as an input, and outputs the number of lines that passes through it.

The rule just checks the length of the list we declared while declaring the stop.

For example, for the centralTrainStation, which is linked to s2, b1 and b2, the output will be 3.

**D - The route calculation:**

This rule is the most complex of the code, it calls a function that calculates the route.

To do that, the tempRoute function checks if both stations are already next to each other, if they are not, it starts the recursive rule.

The recursive rule adds the next stop that is adjacent to Stop1 to the Route, then checks if it is Stop2.

If it is not, it save the new established route as Temp and continues with the next adjacent stop.

If it is, the function exits and return the route. Sadly the route is displayed backwards for the user, so the reverse function reverses it in the main rule before the final display.

**A - The routeTime rule:**

This rule outputs a route specified by the user using the calcRoute rule we just talked about, then calculates the time needed to travel.

For this system, we chose a mean of 6 minutes between 2 stops, which makes 6 minutes per stop minus the one you are already at.

For example, between Airport and Eagle Road, there is 3 stops:

Airport, Sutton Road, and Eagle Road.

As I am already at the Airport, I don’t need to travel there, so it makes 2 stops.

2 stops times 6 minutes gives 12 minutes of traveling.

**D - The route rule:**

This is the main function of the code, the one we executed at the beginning of the presentation.

This rule outputs the shortest and fastest route from a station to another. And, with another parameter, can output the time too.

The rule is using basic prolog implemented rules, it first execute all the possible routes and sort them by time. Then it only shows the first one, being the route with the shortest time.

If two or more routes share the same shortest time, due to the order of the stops declaration, the program will prefer the underground lines first, then the tram lines, and as a last resort, the bus lines.

**Things we could change:**

A - If we wanted to continue working on the program, we could for example:

Add a new dataset of times between each stop to calculate the time more realistically instead of using the same time between each stop.

D - Or for instance add a way for the output to tell at which stop we should switch line, and which one to take. It could be made by use of the sameLine rule inside tempRoute rule.

A - We could also optimize the program to avoid changing line if there is only one stop difference. Because changing mean of transport often loses time.

**Thank you:**

D – Anyway, thank you very much for listening, do you have any questions?