**Rules**

This document contains all the functions used to build the London Tube Navigation Expert System. The Table below is linked to specific sections in the document for easy navigation.

|  |
| --- |
| **Function Name** |
| [Just\_Taking\_Input](#_Rule_–_Just_Taking_Input) |
| [find\_nearest\_attraction\_to\_stations](#_Rule_–_find_nearest_attraction_to_s) |
| [find\_description](#_Rule_–_find_description) |
| [find\_loc](#_Rule_–_find_loc) |
| [same-line](#_Rule_–_same-line) |
| [diff\_line](#_Rule_–_diff_line) |
| [get\_complete\_route](#_Rule_–_get_complete_route) |
| [find-min-length-fact1](#_Rule_–_find-min-length-fact1) |
| [find-min-length-fact](#_Rule_–_find-min-length-fact) |
| [final\_printer\_diff](#_Rule_–_final_printer_diff) |
| [final\_printer\_same](#_Rule_–_final_printer_same) |
| [existing](#_Rule_–_existing) |

## Rule – Just\_Taking\_Input

This rule creates the basic user interface and offers the user different options to interact with the system. The input received is asserted as a special type of fact based on the type of sub-mnu under which input was give. More onformation of the types of facts can be found in the Part II-Group4-Report.docx and the Part II-Group4-text-form-facts-and-templates.docx file submitted.

(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: ")

(bind ?choice (read))

(if (eq ?choice 1) then

(printout t "Enter the first station name: ")

(bind ?station1 (readline))

(printout t "Enter the second station name: ")

(bind ?station2 (readline))

(assert (user-input (name1 ?station1) (name2 ?station2)))

)

(if (eq ?choice 2) then

(printout t "Happy to supply you with more information!" crlf crlf)

(printout t "Choose an option: " crlf crlf)

(printout t "1. Find Nearest Stations to Attraction" crlf crlf)

(printout t "2. Find Information on Attractions" crlf crlf)

(printout t "3. Find Attractions near a Station" crlf crlf)

(printout t "Enter your choice: ")

(bind ?ch (read))

(if (eq ?ch 1) then

(printout t "Enter the name of the attraction: ")

(bind ?attr1 (readline))

(assert (user-input-attr-loc (atr\_name ?attr1)))

)

(if (eq ?ch 2) then

(printout t "Enter the name of the attraction: ")

(bind ?attr2 (readline))

(assert (user-input-attr-det (atr\_name ?attr2)))

)

(if (eq ?ch 3) then (printout t "three" crlf)

(printout t "Enter the name of the Station: ")

(bind ?attr3 (readline))

(assert (user-attr-input (atr\_name ?attr3)))

)

))

## Rule – find\_nearest\_attraction\_to\_stations

This rule looks for input type facts that match stations in the fact base of the expert system and if such a match is found, outputs the nearest attractions.

(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) ", " (nth$ 4 ?st-name) ", " (nth$ 5 ?st-name)) crlf)

)

## Rule – find\_description

This rule looks for input type facts that match attractions in the fact base of the expert system and if such a match is found, outputs the description.

(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)

)

## Rule – find\_loc

This rule looks for input type facts that match attractions in the fact base of the expert system and if such a match is found, outputs the nearest station and tube line.

(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)

)

## Rule – same-line

This rule checks the user input fact and if both start and end stations are on the same line calculates route.

(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))

)

else

(if (member$ ?Station2 ?after1)

then

(bind ?index (find-index ?Station2 ?after1))

(if (= ?index 1)then

(bind ?bp ?bp ?Station2)

else

(bind ?bp ?bp (asplice ?after1 ?index))

)

)

)

(assert(route (Start ?Station1) (End ?Station2)(Station\_list ?bp)(Switch\_list nil) (color\_list ?color) (direct\_flag 1)))

(modify ?f (flag 'TRUE'))

)

## Rule – diff\_line

This rule checks the user input fact and if both start and end stations are on the diiferent line asserts and initializes the route.

(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)))

)

## Rule – get\_complete\_route

This rule calculates the complete route when stations are on a different route by referencing defined functions.

(defrule get\_complete\_route

(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

(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)

then

(bind ?index (find-index (fact-slot-value ?st2 name) ?b1))

(if (= ?index (length$ ?b1))then

;;(bind ?ans (str-cat ?stt (fact-slot-value ?st2 name)))

;;(modify ?r (Station\_list ?ans))

(bind ?tp ?tp (fact-slot-value ?st2 name))

else

(bind ?tp ?tp (bsplice ?b1 ?index))

)

else

(if (member$ (fact-slot-value ?st2 name) ?a1)

then

(bind ?index (find-index (fact-slot-value ?st2 name) ?a1))

(if (= ?index 1)then

(bind ?tp ?tp (fact-slot-value ?st2 name))

else

(bind ?tp ?tp (asplice ?a1 ?index))

)

)

)

)

(bind ?n (+ ?n 1))

)

(modify ?r (Station\_list ?tp))

)

## Rule – find-min-length-fact1

This rule finds the facts with the minimum number of stations and retracts the rest to find the optimal route.

(defrule find-min-length-fact1

(declare (salience 7))

?fact1 <- (route (Station\_list $?slots1) )

?fact2 <- (route (Station\_list $?slots2) )

(test (< (length$ ?slots1) (length$ ?slots2)))

=>

(retract ?fact2)

)

## Rule – find-min-length-fact

This rule finds the facts with the minimum number of line switches and retracts the rest to find the optimal route.

(defrule find-min-length-fact

(declare (salience 6))

?fact1 <- (route (color\_list $?slots1))

?fact2 <- (route (color\_list $?slots2))

(test (< (length$ ?slots1) (length$ ?slots2)))

=>

(retract ?fact2)

)

## Rule – final\_printer\_diff

This rule prints the route for different line stations.

(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

(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)))))

(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)

then

(printout t "Starting from " (fact-slot-value ?st1 name)" take the " (fact-slot-value ?st1 line-color) " line towards " (nth$ 1 ?b1) " and get down at Station " (fact-slot-value ?st2 name) crlf)

else

(if (member$ (fact-slot-value ?st2 name) ?a1)

then

(printout t "Starting from " (fact-slot-value ?st1 name)" take the " (fact-slot-value ?st1 line-color) " line towards " (nth$ (length$ ?a1) ?a1) " and get down at Station " (fact-slot-value ?st2 name) crlf)

)

)

)

(bind ?n (+ ?n 1))

)

(bind ?money (calc\_cost ?st1 ?st2))

;;(modify ?r (price ?money))

(printout t crlf)

(printout t "You will have arrived at your destination" crlf)

(printout t crlf)

(printout t "The total cost of your trip will be: £" ?money crlf crlf)

)

## Rule – final\_printer\_same

This rule prints the route for same line stations.

(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)

then

(printout t "Starting from " (fact-slot-value ?st1 name)" take the " (fact-slot-value ?st1 line-color) " line towards " (nth$ 1 ?b1) " and get down at Station " (fact-slot-value ?st2 name) crlf)

else

(if (member$ (fact-slot-value ?st2 name) ?a1)

then

(printout t "Starting from " (fact-slot-value ?st1 name)" take the " (fact-slot-value ?st1 line-color) " line towards " (nth$ (length$ ?a1) ?a1) " and get down at Station " (fact-slot-value ?st2 name) crlf)

)

)

)

(bind ?money (calc\_cost ?st1 ?st2))

;;(modify ?r (price ?money))

(printout t crlf)

(printout t "You will have arrived at your destination" crlf)

(printout t crlf)

(printout t "The total cost of your trip will be: £" ?money crlf crlf)

)

## Rule – existing

This rule runs when no stations with matching names areund.

(defrule exiting

(declare (salience -1))

(and (not (route)) (user-input))

=>

(printout t "Sorry! No such route exists. Please enter valid Station names." crlf)

)