

**CZ3005 ARTIFICIAL INTELLIGENCE**

**LAB 2 REPORT**

**Subway sandwich interactor**

By: Chiam Jack How U1722261C

Lab Group: TSP4

**School of Computer Science & Engineering**

Overview

The chosen assignment is **Subway sandwich interactor.**

The way the valid ingredients for each category is decided:

If **normal** meal, **all choices** are available.

If **healthy** meal, only **non-fat sauces** will be shown.

If **value** meal, **no top-ups** will be shown.

If **veggie** or **vegan** meal**, no meats** will be shown.

If **vegan** meal, only **non-cheese top-ups** will be shown.

Implementation

For the interface, JavaScript and HTML were used. Prolog was used as the Knowledge Base, however a Prolog interpreter for JavaScript called Tau Prolog (<http://tau-prolog.org/>) had to be used. The website is hosted on <https://rukidevilwalk.github.io/CZ3005-Lab2/>

Prolog was used to populate the contents for the front-end dynamically. The front-end queries Prolog for the next ingredient category to display (e.g. if get\_meats/1 returns nothing then no meat will be shown) and Prolog will render them based on its Knowledge Base.

With well-defined facts and predicates, the Knowledge Base can be easily further populated, and Prolog can be the one populating the front-end. This ensures that minimal modification is needed for the front-end whenever the Knowledge Base changes.

Files:

* Front-end
  + Index.html
  + Index.js
  + Styles.css
* Prolog
  + Core.js
  + Core.pl (same as core.js, but .pl file type. Writes to console instead)
* Prolog Interpreter
  + Tau-prolog.js

**Core.js**

**Declare facts for the Knowledge Base**

% Declare facts for the different types of ingredients

meals([normal, healthy, veggie, vegan, value]).

breads([multigrain, wheat, honey\_oat, italian, parmesan\_oregano, hearty\_italian, flatbread]).

meats([beefsteak, chicken, ham, bacon, beef, meatballs, salmon, tuna, turkey]).

veggies([black\_olives, jalapeno, pickles, cucumber, green\_peppers, red\_onions, tomatoes, lettuce]).

fat\_sauces([chipotle, bbq, ranch, sweet\_chilli, mayo]).

non\_fat\_sauces([honey\_mustard, sweet\_onion, yelow\_mustard, deli\_brown\_mustard]).

non\_vegan\_topups([american, monterey\_cheddar, processed\_cheddar]).

vegan\_topups([avocado, egg\_mayo]).

sides([yogurt, chips, cookies, hashbrowns, drinks]).

% Declare fact for different types of behaviours

behaviour([tired\_behaviour, energetic\_behaviour, calm\_behaviour]).

% Declare facts for the different types of staff's behaviour

behaviour(tired\_behaviour, [closing\_eyes, grouchy\_look, listless\_eyes]).

behaviour(energetic\_behaviour, [broad\_smile, beaming\_voice, whistling]).

behaviour(calm\_behaviour, [looking\_composed, looking\_attentive, light\_smile]).

% Declare facts for meal types, used for checking the valid choices

is\_healthy\_meal(healthy).

is\_value\_meal(value).

is\_vegan\_meal(vegan).

is\_veggie\_meal(veggie).

**Declare predicates for retrieving the valid choices for each ingredient category. For meats, sauces and top-ups categories, the valid choices are based on the type of meal was asserted. The type of meal is checked through is\_healthy\_meal(healthy), is\_value\_meal(value), is\_vegan\_meal(vegan) and is\_veggie\_meal(veggie) facts that had been established.**

% Declare predicates for getting the valid ingredients based on current arguments

% e.g get\_meats(X) will return empty if the selected meal type is veggie

get\_meals(X) :- meals(X).

get\_breads(X) :- breads(X).

get\_meats(X) :-  % No meat for vegan/veggie meal types

findall(X, (selected\_meals(Y), \\+is\_veggie\_meal(Y), \\+is\_vegan\_meal(Y), meats(X)), X).

get\_veggies(X) :- veggies(X).

get\_sauces(X) :-  % No sauces for healthy meal type

findall(X, (selected\_meals(Y), is\_healthy\_meal(Y) -> non\_fat\_sauces(X);

non\_fat\_sauces(L1),  fat\_sauces(L2), append(L1, L2, X)), X).

get\_topups(X) :-  % No topup for value meal type; No cheese topup for vegan meal type

findall(X, (selected\_meals(Y), \\+is\_value\_meal(Y) -> (is\_vegan\_meal(Y) -> vegan\_topups(X);

non\_vegan\_topups(L1), vegan\_topups(L2), append(L1, L2, X))), X).

get\_sides(X) :- sides(X).

**selected/2 is used to assert facts only if the input is not already asserted which is done through check\_selection/2**

% Declare predicates for asserting facts if given input is not already selected

% only will assert if X is not in the selected list e.g check\_selection(X, meals) is false

selected(X,meals) :-

 \+check\_selection(X, meals) -> asserta(selected\_meals(X)).

selected(X,breads) :-

\+check\_selection(X, breads) -> asserta(selected\_breads(X)).

selected(X,meats) :-

\+check\_selection(X, meats) ->asserta(selected\_meats(X)).

**.**

**.**

**.**

% Declare predicates for checking if X is already in selected list

check\_selection(X, breads):-

selected\_breads(L), member(X,L),!.

check\_selection(X, meats):-

selected\_meats(L), member(X,L),!.

check\_selection(X, meals):-

selected\_meals(L), member(X,L),!.

**.**

**.**

**.**

**GUI**

**For every new order setStaffBehaviour/1 is used to select a random behavior using random\_member/2 from behavior/2 which was established earlier on, and a random gesture from that behavior is selected for the opening message from the subway staff. The HTML DOM element is then created for that gesture through displayStaffGesture/1.**

% Chooses a random behaviour

% Then chooses a random gesture assigned to that selected behaviour

setStaffBehaviour(list) :-

behaviour(List1),

random\_member(B1, List1),

behaviour(B1, List2),

random\_member(B, List2),

displayStaffGesture(B).

% Display staff gesture

displayStaffGesture(B) :-

create(a, A),

    atom\_concat('-', B, Y),

    atom\_concat(Y, '- Welcome to Subway, what kind of meal would you like?', Z),

    html(A, Z),

    get\_by\_id('subway-header', Parent),

    append\_child(Parent, A).

**For each ingredient category, options/1 is used to retrieve the valid choices for each category based on the meal type selected. createDOMV1/1 and createDOMV2/2 are called recursively to create the list of menu items and buttons for the ingredients through calling createMenuItems/1 and createButton/1 which creates the HTML DOM elements.**

% Declare predicates for getting the menu item list based on current arguments and creates a menu for GUI

options(meals) :-

get\_meals(L),

createDOMV1(L).

options(sauces) :-

get\_sauces(L),

createDOMV2(L).

options(breads) :-

get\_breads(L),

createDOMV1(L).

options(meats) :-

**.**

**.**

**.**

% CreateDOMV1 is used to create the HTML DOM for the front end based on current list

% Uses createButton(H) and createMenuItems(H)

% createDOMV1 and createDOMV2 is called recursively until the list is exhausted

createDOMV1([]). % empty list

createDOMV1([H]) :- % last item in list

createButton(H),

createMenuItems(H).

createDOMV1([H|T]) :-  % List with items more than one

createButton(H),

createMenuItems(H),

createDOMV1(T), !.

% createDOMV2 is the same as createDOMV1 except it's for nested lists

createDOMV2([[]]). % empty list

createDOMV2([[H]]) :- % last item in list

createButton(H),

createMenuItems(H).

% List contains more than 1 item

createDOMV2([[H|T]]) :-

createButton(H),

createMenuItems(H),

createDOMV2([T]), !.

% Create menu items for each ingredient category in the GUI

createMenuItems(H) :-

create(a, A),

    html(A, H),

create(br, BR),

    get\_by\_id('subway-contents', Parent),

    append\_child(Parent, BR),

    append\_child(Parent, A).

% Create a button for each ingredient in each category in the GUI

createButton(H) :-

create(button, BUTTON),

add\_class(BUTTON, 'btn btn-outline-success btn-sm'),

    set\_attr(BUTTON,type, button),

    set\_attr(BUTTON,value, H),

    html(BUTTON, H),

    get\_by\_id('btn-group', Parent),

    append\_child(Parent, BUTTON).

**displaySelections/1 will create the final order for selected ingredients. Predicates, e.g show\_meals/1, are used to retrieve all selected ingredients and displayOrder/2 is called recursively to create the HTML DOM for display.**

% Declare predicate for displaying all the selected ingredients for the final order on the GUI if input X is 1

displaySelections(X) :-

(X==1) ->

show\_meals(Meals),

show\_breads(Breads),

show\_meats(Meats),

**.**

**.**

% Declare predicates for for getting the corresponding choices based on user's input

% e.g findall(X, pred(X), List) - Finds possible values for predicate and displays them on the GUI

show\_meals(Meals) :-

findall(X, selected\_meals(X), Meals),

displayOrder('Meal:',Meals).

show\_breads(Breads) :-

findall(X, selected\_breads(X), Breads),

displayOrder('Breads: ',Breads).

show\_meats(Meats) :-

findall(X, selected\_meats(X), Meats),

displayOrder('Meats: ',Meats).

**.**

**.**

**.**

% For displaying the final order

% Used to add the selected ingredients to a <a></a> and appends to the GUI

% displayOrder is called recursively until the list is exhausted

displayOrder(X,[]):- % empty list

create(a, A),

atom\_concat(X, 'None.', Y),

    html(A, Y),

create(br, BR),

    get\_by\_id('subway-contents', Parent),

    append\_child(Parent, A),

    append\_child(Parent, BR).

displayOrder(X,[H]) :- % last item in list

create(a, A),

atom\_concat(H, '.', Y),

    html(A, Y),

create(br, BR),

    get\_by\_id('subway-contents', Parent),

    append\_child(Parent, A),

    append\_child(Parent, BR).

displayOrder(X,[H|T]) :-  % List with items more than one

create(a, A),

atom\_concat(H, ', ', Y),

    html(A, Y),

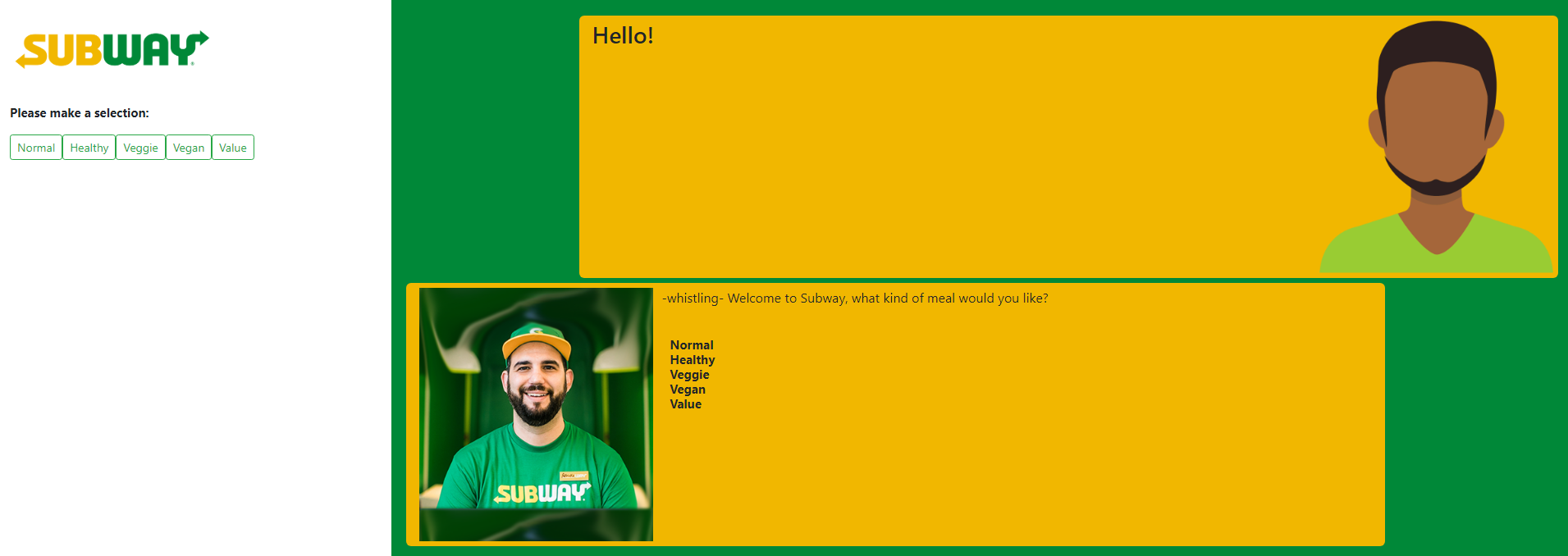
    get\_by\_id('subway-contents', Parent),

    append\_child(Parent, A),

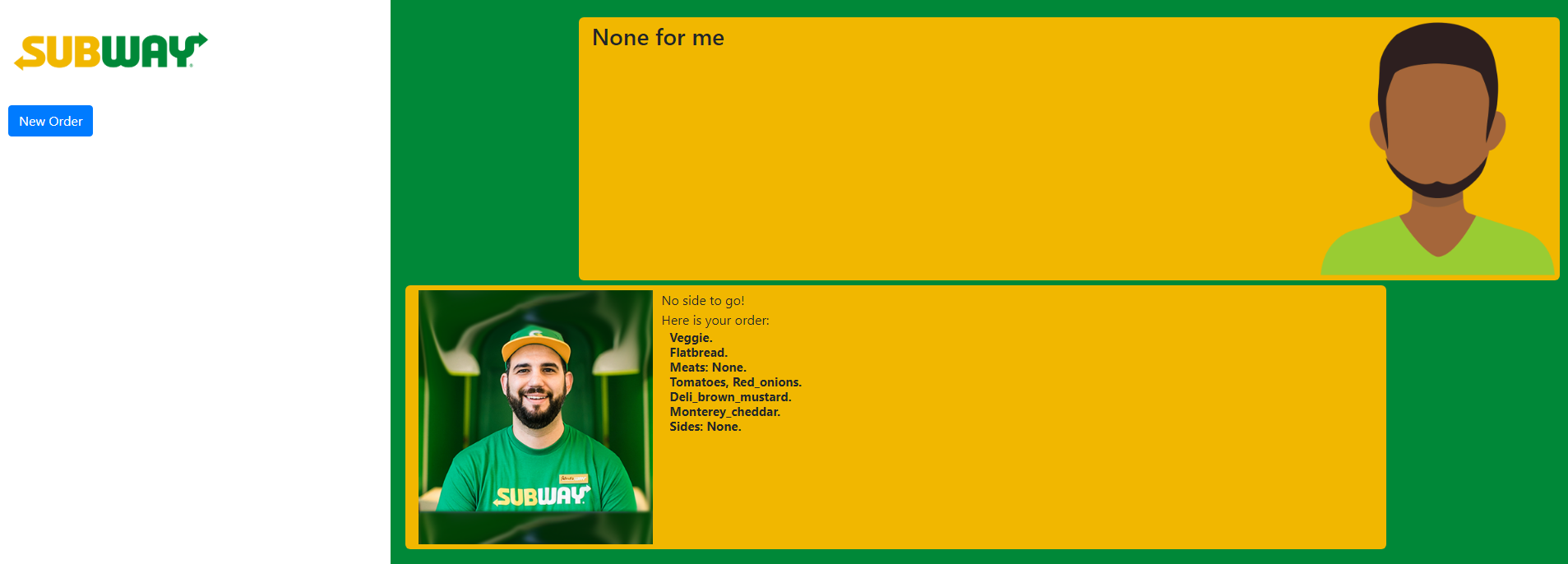
displayOrder(X,T), !.

**GUI Demo Example**

Select ingredients by clicking the buttons:



Order Completed:



**Core.pl**

**CLI version Demo**

options(meats) returns false as veggie meal was selected

*Note: there is no formatting as the choices are meant to be displayed on the front-end GUI*

