NANYANG TECHNOLOGICAL UNIVERSITY

CZ3005 ARTIFICIAL INTELLIGENCE

ASSIGNMENT 2 REPORT

HENG WEILIANG

U1620593L

TS3

Assignment 3: Subway sandwich interactor

In this assignment, a Prolog-based interactor would be implemented to mimic the Subway sandwich ordering experience. Customer would be able to customize their order from a predefined range of options.

The implementation is split into the core Prolog script and a GUI in QT and some of the ordering logica in python. `Subway.pl` stores the Subway model in Prolog and `Subway.py` store the model of the system and some of the actions to be perform. The files `qt4.py` and `qt5.py` are the UI interface implemented in QT4 and QT5. Both qt4.py and qt5.py could be used to start application.

Model: Subway.pl

The model in subway.pl includes three dynamic predicate `order/1, reject/1` and `pass/1`. The predicate `order/1` store a list of item that the customer has order. The predicate `reject/1` store a list of item that the customer has choose not to include in the meal. The predicate `pass/1` is a special list which store the rest of the bread.

```
askBread(X, Y):- bread(L), member(X,L).
askMains(X):- mains(L),member(X,L).
askCheeses(X):- cheeses(L),member(X,L).
askSauces(X):- sauces(L),member(X,L).
askVegs(X):- vegs(L),member(X,L).
askCookies(X):- cookies(L),member(X,L).
askAddons(X):- addons(L),member(X,L).

:- dynamic order(nothing).
:- dynamic reject(nothing).
:- dynamic pass(nothing).
```

The model also includes 7 predicate to provide the next item in each category. They are `askBread/2, askMains/1, askCheeses/1, askSauces/1, askVegs/1, askCookies/1 and askAddons/1.` They all depend on the predicate `member/2` which is a recursive predicate to extract an item from the list. I left the predicate `out/2` and `add/2` in the file as they are used initially to add item to list and takeout item from list.

Python Model: Subway.py

The model written in python is mainly use to define some methods that the system may offer. I model the `Subway.pl` into a python class `Subway`.
The class is initialised with several attributes mostly the predicate and attributes to determine what is in the predefine meal options.

```
    def __init__(self):
        # super().__init__()
        self.mains = Functor('askMains', 1)
        self.cheeses = Functor('askCheeses', 1)
        self.vegs = Functor('askVegs', 1)
        self.sauces = Functor('askSauces', 1)
        self.cookies = Functor('askCookies', 1)
        self.addons = Functor('askAddons', 1)
        self.func = self.vegs
        self.selectMain = True
        self.selectCheeses = True
        self.selectVegs = True
        self.selectSauces = True
        self.selectCookies = True
        self.selectAddons = True
        self.prolog = Prolog()
        self.prolog.consult("Subway.pl")
```

The methods `setVeggie` and `setVegan` is used to determine what are the category that are offered in the predefine meal options.

```
    def setVeggie(self):
        self.selectMain = False
        self.selectCheeses = False

    def setVegan(self):
        self.selectMain = False
        self.selectCheeses = False
        self.selectCookies = False
```

The methods `askBread`, `yesBread` and `noBread` is called when customer select a bread or reject a bread. In `yesBread` the special predicate `pass/1` is used.

```
	def askBread(self):
		n = list(self.prolog.query('askBread(X,Y)',
maxresult=1))[0]['X']
		return n

	def yesBread(self, y):
		self.prolog.asserta('order(' + y + ')')
		for i in list(self.prolog.query('askBread(X,' + y + ')')):
			if i['X'] != y:
				self.prolog.asserta('pass(' + i['X'] + ')')

		n = self.Event()
		return n

	def noBread(self, y):
		self.prolog.asserta('reject(' + y + ')')
		n = list(self.prolog.query('askBread(X,' + y + ')'))
		return n
```

The method `nextBread` is used to generate the next kind of bread to offer by calling the predicate `askBread/2` and comparing with the predicate `reject/1`.

```
	def nextBread(self, y):
		n = self.reject()
		for x in list(self.prolog.query('askBread(X,' + y + ')')):
			if x['X'] not in n:
				return x['X']
```

The `Event` method is the analogous to the item to be offered after a bread is selected. It makes use of the `deterFunc` method ask call a particular predicate define in the Subway.pl.

```
	def Event(self):
		self.deterFunc()
		s = []
		a = []
		x = Variable()
```

```
            q = Query(self.func(x))
            while q.nextSolution():
                  s.append(x.value)
            for i in s:
                  a.append(i.value)
            n = a[0]
            q.closeQuery()
            return n
```

Both `yesEvent` and `noEvent` is to mimic the behaviour of asserting a predicate into the current knowledge base. Thus both `order/1` and `reject/1` is defined as dynamic predicate.

```
      def yesEvent(self, y):
            self.prolog.asserta('order(' + y + ')')


      def noEvent(self, y):
            self.prolog.asserta('reject(' + y + ')')
```

Both doneItem and deterFunc methods is used to determine the current and next predicate to be called. This is achieved by looking at the current predicate executing and and trigger when the customer choose to continue to next category of items.

```
      def doneItem(self):
            if self.func is self.mains:
                  self.selectMain = False
            elif self.func is self.cheeses:
                  self.selectCheeses = False
            elif self.func is self.vegs:
                  self.selectVegs = False
            elif self.func is self.sauces:
                  self.selectSauces = False
            elif self.func is self.cookies:
                  self.selectCookies = False
            elif self.func is self.addons:
                  self.selectAddons = False

      def deterFunc(self):
            if self.selectMain:
                  self.func = self.mains
            elif self.selectCheeses:
```

```
            self.func = self.cheeses
        elif self.selectVegs:
            self.func = self.vegs
        elif self.selectSauces:
            self.func = self.sauces
        elif self.selectCookies:
            self.func = self.cookies
        elif self.selectAddons:
            self.func = self.addons
```

The method `nextToOffer` is called when customer are done with choose an item either yes or no. It generates the next item to be offered by executing the current ask predicate.

```
    def nextToOffer(self, y):
        n = self.order() + self.reject()
        s = []
        a = []
        self.deterFunc()

        x = Variable()
        q = Query(self.func(x))
        while q.nextSolution():
            s.append(x.value)
        for i in s:
            a.append(i.value)

        q.closeQuery()
        for x in a:
            if x not in n:
                return x
```