Rapport de Projet :
Le conseiller en vins.

Groupe 7
Julien Albert
Michel Caluwaerts
Philippe Leroy

UNIVERSITÉ DE NAMUR

7 janvier 2018

Table des matières

1 Présentation générale du programme

Notre programme est inspiré du code Eliza, que nous avons simplifié. L'objectif que nous nous étions fixés pour pour la remise comprenait les fonctions suivantes :

- Récupérer les principaux attributs d'un vin donné (bouche, nez, région, description et prix)
- Récupérer une liste de vins via certains critères (prix et région),
- Réaliser les associations plat-vin dans les deux sens,
- Mémoriser la dernière question et pouvoir la rappeler immédiatement, pour obtenir soit plus d'attributs, soit une plus longue liste de vins,
- proposer un concept pour enrichir la base de connaissance par l'utilisateur.
- développer, au fur et à mesure de la croissance du code, une fonction permettant de re-tester les règles précédentes pour prévenir de possibles effets de bord des nouveaux changements.

Du point de vue modularité, nous avons choisi de localiser notre implémentation dans `produire_reponse/2` (hormis les variables globales nécessaires à la mémorisation). Il nous est également paru utile de regrouper les fonctions de même type dans des fichiers séparés.

On peut représenter en pseudo-code le traitement effectué par `produire_reponse(Question, Réponse)` :

```
forall(mots in Question) {
    traduire synonymes();
    }

forall(mots in Question_simplifee) {
    trouver les mots cle;
    if (mots cle = {})
            {mots cle = {notfound}}
    }

trier mots clé();

forall(mot clé in mots cle tries) {
    tester règle r tel que index(r) = mot clé ;
            if(match (Question, Pattern) && réponse trouvée) {
                Réponse = Réponse(r)}
    }
```

La contrainte, pour associer le plus correctement la bonne règle, est triple :

1. être capable d'identifier la question à un pattern de question,
2. être capable de récupérer les variables dans la question,
3. activer un "cas limite" pour une formulation qui ne correspond à aucun pattern.

La section 2 détaille respectivement la base de connaissances, la gestion des mots-clé, les mécanismes d'unification à une règle, de mémorisation d'une question sur une relance de type "en avez-vous d'autres ?", la complétion de la base de connaissance par interrogation de l'utilisateur et le module de test. La section 3 détaille les choix de regroupement des fichiers. Ensuite la section 4 détaille les outils utilisés et le planning suivi. Finalement, la section 5 propose un retour d'expérience en guise de conclusion.

2 Définition des modules

2.1 Base de connaissances

La première tâche à effectuer était de créer la base de connaissances.

Nous avons analysé, en plus de l'énoncé, plusieurs sites présentant des vins, pour en déduire différents attributs intéressants (bouche, nez, prix, région et description). C'est sur le site de Grafé Lecoq que nous avons trouvé les informations nécessaires pour établir une base de connaissances presque complète d'une vingtaine de vins, que nous avons intégrée par copier/coller dans un fichier excel.

L'étape suivante était de déterminer la forme des prédicats nécessaires (liste simple, liste de listes...). Nous avons ensuite créé un programme en Delphi, pour lire le fichier excel, adapter les données, et créer un fichier Prolog avec les prédicats. De cette manière, toute modification du tableau excel pouvait être propagée en quelques secondes au fichier Prolog. Le code source du programme en Delphi est en annexe.

Nous avons envisagé d'utiliser une base de donnée externe (type sql) au lieu de coder "en dur" les prédicats dans le code Prolog. Nous n'avons pas retenu cette solution car il aurait été nécessaire, lors de la remise du code, de remettre une copie de la base de données qu'il aurait fallu réinstaller.

2.2 Mots-clé

Nous avons commencé par établir une liste de questions susceptibles d'être posées, réparties selon les différentes catégories ci-dessous :

- questions simples de type : "quelle est la valeur de l'attribut de tel vin ? " (bouche, nez, région, prix, description),
- questions simples de type : "quels sont les vins correspondant à tel critère ? " (prix ou région),
- questions relatives aux associations plat-vin (dans les deux sens),
- questions sur des résultats précédents (mémorisation).

Cette liste nous a permis de mettre en évidence une série de mots-clé.

Parmi ces derniers, nous avons choisi un mot-clé canonique par type de questions afin de limiter le nombre de règles. Ensuite, nous avons mis en place des règles de substitution pour traiter les mots-clé non retenus (via les prédicats `simplify` et `sr`). Notre chatbot prend également en compte certaines variantes orthographiques, mais pas de manière systématique. Une amélioration pourrait d'ailleurs être de le faire en implémentant des prédicats spécifiques basés sur des expressions régulières [1]. Finalement, nous avons attribué des poids plus élevés pour les mots les plus significatifs afin d'arbitrer les conflits. (Vous trouverez à la fin de ce document un tableau récapitulatif des questions et mots-clé associés.)

2.3 Règles d'association

Le format d'une règle est le suivant, et toutes sont basées sur le même pattern :

```
regle( [motclé, poids], [ID, [Pattern_de_question], Count, Réponse]], Question) :-
                match(Question,Pattern_de_question,
                autres_conditions.
```

Dans certains cas, un mot clé explicite (tel que 'bouche' ou 'nez') suivi quelque part d'un nom de vin doit suffire à suggérer la réponse. Dans ce cas, on va chercher à relâcher au maximum la

contrainte d'identification au pattern. Dans d'autres cas, on aura besoin d'être capable de resserrer le sens de la question et de capter l'ordre dans lequel les variables sont données. (ex : '...(un vin) entre X et Y'). Dans ces cas, nous avons un prédicat `match(Question, Pattern)`, inspiré de [2] qui nous permet de contraindre plus fortement la syntaxe de la phrase, et de récupérer les variables qui nous intéressent. En combinant les deux, nous parvenons à avoir un mécanisme assez flexible de reconnaissance des questions, sans avoir à multiplier les règles avec toutes les variantes syntaxiques.

Nous avons créé les règles de manière itérative à partir des règles les plus simples (renvoyant des réponses descriptives). Par conséquent, elles sont groupées par type de question, dans des fichiers `r1_*`, `r2_*`, `r3_*` et `r4_*`. Nous avons prévu dans chaque groupe de règles les cas limite :

- un mot clé est identifié, mais la question ne matche avec aucun pattern de règle
- la question est unifiable à un pattern, mais la base de connaissance ne peut y trouver de réponse,
- pour les règles utilisant des entiers : les variables fournies ne sont pas du type entier.
- enfin, en l'absence de mot-clé identifié dans la question, une règle générique remonte un message d'erreur.

2.4 Mémorisation

La mémorisation nous sert pour deux cas particuliers :

Le premier cas consiste à afficher une liste partielle de vins. Quand l'utilisateur pose une question dont la réponse est une liste de vins, Grandgousier en propose seulement trois, pour ne pas surcharger l'affichage. Si l'utilisateur demande d'afficher d'autres vins (Q : "en avez vous d'autres ?"), les trois vins suivants seront proposés, et ainsi de suite jusqu'à signaler qu'il n'y en a plus. Cette technique nécessite de mémoriser la liste des vins correspondant à la question posée, ce que nous faisons dans une variable globale `liste_memo`. A chaque affichage d'une série de 3 vins, nous supprimons ceux-ci de `liste_memo`.

Le second cas intervient quand l'utilisateur pose une question sur un vin particulier. Grandgousier va donner en première réponse l'attribut du vin demandé. Si l'utilisateur demande à en savoir plus (Q : "pouvez-vous m'en dire davantage ?"), Grandgousier donnera un autre attribut, et ainsi de suite jusqu'à avoir épuisé les attributs disponibles. Ici encore, la mémorisation est nécessaire. Nous utilisons la variable globale `vin_memo` qui contient l'identifiant du vin, ainsi que ses attributs pas encore cités. A chaque fois que Grandgousier affiche un attribut, il retire cet attribut de la liste `vin_memo`.

2.5 Apprentissage de faits

Nous avons prévu d'ajouter un prédicat d'apprentissage de réponses, à activer quand un ID vin est détecté, et une règle unifiée, mais le contenu de la réponse ne se trouve pas dans la base de donnée.

Le plus complexe en Prolog nous a semblé être la récupération de la saisie et le formatage des phrases en liste de listes. Nous avons donc simplement réutilisé l'utilitaire fourni par M. Jacquet. La saisie de l'utilisateur à une question de type `attribut/2` sur l'ID `Vin_x` va ajouter via `assert`, un fait de type `attribut(Vin_x, [input utilisateur])` à la base de connaissance. Nous l'avons implémenté de manière conceptuelle sur les prédicats `bouche/2` et `nez/2` qui fournissent des descriptions qui sont plus difficilement exploitables. Mais le principe peut très bien fonctionner de manière utile pour ajouter des faits tels qu'une appréciation cotée, un nouveau nom de vin à suggérer, etc...

2.6 Tests unitaires

Afin de vérifier de manière plus systématique le bon fonctionnement du chatbot, nous avons implémenté un module de tests unitaires. Celui-ci permet de vérifier d'une part qu'un input correct (une question sous forme d'une liste de mot) ne génère pas un message d'erreur et d'autre part qu'un input incorrect génère bien un message d'erreur. Il est activé via le prédicat `test_programme`.

Outre le debugging proprement dit et la détection des régressions lors des mises à jour, ce module permet également, avec l'utilisation de prédicats d'impression, de vérifier la bonne application des règles et la pondération des mots-clé. Cependant, étant stateless, il n'est pas utilisable pour les règles liées à la mémorisation.

Ce module reste relativement simple et ne permet pas de tester finement les inputs. Pouvoir écrire des tests de la forme `test([input_souhaite], [output_souhaite]` serait d'ailleurs une amélioration substantielle pour la vérification du code. L'utilisation d'un framework dédié [3] pourrait être également un plus.

3 Organisation des fichiers

Afin de faciliter la compréhension, l'implémentation et la maintenance de notre code, celui-ci a été réparti dans différents fichiers de la manière suivante :

- `grandgousier.pl`.
- `base_connaissance.pl`.
- `r0_autres_regles.pl` : règles non business, dont `notfound`.
- `r1_questions_simples1.pl` : règles relatives aux questions simples de type : "quelle est la valeur de l'attribut de tel vin ?".
- `r2_questions_simples2.pl` : règles relatives aux questions simples de type : "quels sont les vins correspondant à tel critère (prix ou région) ?".
- `r3_questions_accompagnement.pl` : règles relatives aux associations plat-vin.
- `r4_memorisation.pl` : règles relatives à la mémorisation.
- `r5_learning.pl` : règles relatives l'apprentissage.
- `tests_unitaires.pl`.
- `util_affichage.pl` : mise en forme des réponses.
- `util_conversion.pl` : conversion des questions en listes de mots et inversement.
- `util_liste_mots.pl` : règles de simplifications, mots-clés et outils [4].
- `util_memorisation.pl` : gestion des variables globales liées à la mémorisation.
- `util_recherche.pl` : recherche de résultats (listes de vins ou de plats).

Même si cette tactique permet de séparer dans une certain mesure les différentes parties du chatbot, le couplage entre les fichiers reste très important. Par exemple, la question de la mise en forme de la réponse est traitée spécifiquement dans `util_affichage.pl` mais aussi dans une partie des règles (fichiers `r*`, ce qui complique l'amélioration de cette partie et dans une moindre mesure l'écriture des tests. Une piste d'amélioration serait donc de modulariser davantage notre code afin de diminuer le couplage entre les parties et donc de faciliter l'évolution de celles-ci.

Une autre piste d'amélioration, davantage axée sur l'écriture du code et des spécifications, serait de fixer et de suivre des règles de bonne pratique, par exemple [5].

4 Méthodologie de travail

4.1 Outils utilisés

4.1.1 Interpreteur Prolog

Philippe utilise le module Prolog d'Eclipse. Michel utilise SWI-Prolog en console, ainsi que Swish, la version web, qui apporte un vrai confort pour tester un bout de code à la volée, ainsi qu'une belle interface pour la fonction `trace`. Julien utilise uniquement SWI-Prolog en console.

4.1.2 Repository

Un dépôt privé a été créé sur GitHub afin de gérer le versionning du code. Étant donné le peu de temps à disposition, nous avons préféré travailler uniquement sur la branche principale. Dans cette même optique, la consigne était que chaque modification n'entraîne pas de régression.

4.1.3 Génération des prédicats

Comme dit précédemment, nous avons créé un programme Delphi afin de générer les faits de la base de connaissance à partir d'un fichier excel.

4.2 Planning

Le planning initial prévoyait la livraison de la dernière itération pour le 22 décembre, ce qui n'a pas été tenu. Le planning réalisé au final aura été le suivant :

- 4 déc. : première version compilable du `grandgousier.pl` avec des règles de test.
- 19 déc. : génération automatique des prédicats de la DB.
- 20 déc. : livraison des règles simples dans fichier `r1_*` (règles descriptives).
- 22 déc. : fichier de règles `r2_*` (prix, liste par région).
- 28 déc. : règles d'association plat-vin.
- 29 déc. : module apprentissage et tests unitaires.
- 31 déc. : fonction mémorisation, debugging et version opérationnelle finale.

5 Retour d'expérience

- Pour un premier exercice de programmation modulaire en Prolog, le recours à la fonction `trace` est relativement plus utile et plus fréquent que dans les langages appris précédemment, du fait que l'interface de Prolog est moins verbeuse.
- Nous avons procédé de manière itérative, mais il aurait été possible de pousser plus loin la réflexion sur les modules du code avant de procéder à son implémentation. Certains refactorings ont d'ailleurs été nécessaires.
- Prolog est un langage sur lequel on trouve encore très facilement des ressources, tutoriels, forums de discussion. C'est un paramètre qui en facilite l'accès.

Références

[1] M. Hendricks, "Package regex," http ://www.swi-prolog.org/pack/list ?p=regex, consulté le 2018-01-07.

[2] D. Merritt, "Dr. Dobb's AI Expert Newsletter, issue #111," 2003. [Online]. Available : (via)https://trilug.org/pipermail/dev/2003-September/000368.html

[3] J. Wielemaker, "Prolog unit tests," http ://www.swi-prolog.org/pldoc/doc_for ?object=section(consulté le 2018-01-07.

[4] P. Blackburn, J. Bos, and K. Streignitz, *Prolog, tout de suite !* College Publications, 2007.

[5] M. A. Covington, R. Bagnara, R. A. O'Keefe, J. Wielemaker, and S. Price, "Coding guidelines for prolog," *CoRR*, vol. abs/0911.2899, 2009. [Online]. Available : http://arxiv.org/abs/0911.2899

Questions	**Mots-clé (poids)**	**Variantes**
Que donne le [vin] en bouche ?	bouche (10)	
Comment est l'attaque du [vin] ?		attaque
Parlez-moi de la longueur du [vin] ?		longueur
Quel nez présente le [vin] ?	nez (10)	
Quels sont les arômes du [vin] ?		arômes
Quel est le bouquet du [vin] ?		bouquet
Quelles sont les propriétés olfactives du [vin] ?		olfactives
Quelle est l'appellation du [vin] ?		appellation
De quelle région provient le [vin] ?	region (4)	
D'où provient le [vin] ?		provient
De quelle origine est le [vin] ?		origine
Quel est le prix du [vin] ?	prix (7)	
Combien coûte le [vin] ?		coute
À combien puis-je acheter le [vin] ?		acheter
Pouvez-vous me parler du [vin] ?		parler
Décrivez-moi le [vin] ?	description (9)	décrivez
Présentez-moi le [vin] ?		présentez
Que pouvez-vous me dire au sujet du [vin] ?		dire
Avez-vous des vins de la région [région] ?	région (4)	
Avez-vous des vins de [région]		avez, auriez, conseille(r/z)
Auriez-vous des vins entre [prix min] et [prix max] ?	entre (10)	fourchette
Quel est votre vin le plus/moins cher ?	cher (9)	onéreux
Auriez-vous des vins à moins de [prix max] ?	moins (8)	
Auriez-vous des vins à plus de [prix min] ?	plus (8)	
Quel vin pour accompagner du [nom plat] ?	accompagner (8)	
Quel vin irait avec du [nom plat] ?		aller
Je vais cuisiner du [nom plat], que conseillez-vous ?		cuisiner
Quel plat pour accompagner du [vin] ?	plat (9)	plats, met(s), préparation(s)
Avez-vous davantage d'information ?	davantage (10)	
Avez-vous davantage d'information sur [vin] ?		
En avez-vous d'autres ?	autre (10)	(d)autre(s)

TABLE 1 – Récapitulatif des questions et des mots-clé associés