24/12/2014

[IA01] Système expert

Compte-rendu TP n°3

HENRY LEE
ETIENNE LAVIOLETTE

Table des matières :

Introduction : 2
I. Synthèse et expertise : 3
II. Interaction Homme-Machine et liste des sports 9
III. Moteur d'inférence et fonctions de service 20
IV. Limites et ambitions futures de développement : 26
Conclusion : 28
Annexe : 29

Introduction :

Le sujet de ce dernier TP du semestre A14 d'IA01 est de réaliser un système expert d'ordre 0+.

L'objectif du présent rapport sera de présenter le système expert d'ordre 0+ de manière générale afin, par la suite, d'expliciter au mieux le sujet que nous aurons choisi ainsi que les sources d'expertises qui sont tout à faits primordiales.

Le polycopié ainsi que les cours d'IA01 dispensés par Mme. Abel et Mr.Fontaine nous ont permis de comprendre l'utilité des systèmes experts quant à la résolution de problèmes difficiles. Ceux-ci nécessitant la plupart du temps l'analyse d'un expert humain. Le but de cette branche de l'intelligence artificielle est, si ce n'est de remplacer un expert dans sa globalité, de l'aider dans sa démarche d'expertise. C'est donc un travail, main dans la main avec les connaisseurs.

La base d'un système expert est un triptyque composé :

- D'une base de faits, la mémoire à court terme de notre programme,
- D'une base de règles constituant les connaissances à long terme de notre sujet,
- Un moteur d'inférence confrontant les deux entités citées précédemment afin d'avancer dans la résolution de notre problème

Cependant tous les systèmes experts ne sont pas équivalents. Trois classes différents existent et nous ont été présentées :

- Le système expert d'ordre 0 constitué de règle à valeur de vérité booléenne *: Si y alors z.*
- Le système expert d'ordre 0+ qui nous intéressera ici, remplace les valeurs booléennes par des couples « attribut-valeur » : *Si x=4 alors z =0*
- Le dernier type de système expert, pour sa part, fait intervenir des variables ($).

Les systèmes d'ordre 0 et 1 seront laissés de côté dans notre étude puisque nous nous intéresserons dans la suite du TP à la réaliser d'un système expert faisant intervenir des attributs et leur valeur.

I. Synthèse et expertise :

Synthèse du retour de Mme Abel :

Lorsque nous avons proposé notre idée de système expert ainsi que l'introduction demandée pour la mi-décembre à Mme. Abel nous avons reçu comme retour que nous devions intégrer des règles autre que celle de l'égalité dans la construction de notre base de règle pour ainsi nous retrouver en présence d'un système expert d'ordre 0+.

Les deux sujets que nous avons proposés à ce moment précis étaient les suivants :

- Représentation du jeu Jumanji en système expert par l'intermédiaire d'un jeu d'aventure. Un problème a vite été soulevé par un collègue avec qui je discute régulièrement d'IA à l'UTC. En effet ce sujet est un des sujets présents sur internet comme « annale » de projet n°3 de l'UV IA01. Ne voulant pas être accusés de fraude ou de triche nous avons décidé de ne pas sélectionner ce sujet.

- Il nous restait alors notre seconde idée : un système expert nous aidant à la décision sur le choix d'une tenue en fonction de diverses circonstances arbitraires que nous aurions choisies. Notre source d'expertise étant toute trouvée puisque nous avions un contact créateur d'une marque de mode masculine à Paris pour nous aider à construire nos règles et articuler les différentes parties de notre sujet. Au moment de contacter mon ami Geoffrey Bruyère, j'ai eu la désagréable surprise d'apprendre que celui-ci se lançait de Décembre à Janvier dans une traversée de l'Europe d'Ouest en Est : de Paris à Istanbul en vélo dans le but d'étendre sa culture et de se rendre, par un moyen qu'il adore, dans ce pays réputé pour ses tissus nobles. Son ami et acolyte Benoit Wojtenka étant lui parti au pays du soleil levant pour du « sourcing » (repérage de bon manufacturier peu connu pour leur prochaine collection de collaborations). Henry ne connaissant pas de sources dans ce domaine nous nous sommes retrouvés face à un dilemme assez embêtant.

Après plusieurs jours de réflexions, au vu des remarques faites par Mme. Abel et au vu de notre problème de dernière minute nous avons décidé de changer de domaine d'expertise. Ayant déjà rencontré Henry au Badminton il nous est apparu évident que le sport constituerait un domaine d'expertise privilégié. Pratiquant tous les deux le sport intensément (de nombreuses heures de badminton pour Henri et plus d'une dizaine d'année de football et 5 ans de badminton ainsi que de nombreux autres sports pour ma part), nous avons décidé de nous réorienter.

Cependant, tout le travail de recherche de sources, d'édification des différentes bases, de cohérence de celles-ci était à refaire. Nous nous sommes donc d'abord mis en quête de sources fiables et pouvant nous apporter une expertise intéressante.

Au vu du nombre de sportif d'élite et de sports enseignés à l'UTC ainsi qu'à Compiègne nous n'avons pas mis longtemps pour rassembler de nombreuses informations sur ce sujet.

Notre problématique était assez claire. Ayant été marqués par le système MyCin que vous nous avez présenté en cours de semestre nous nous sommes tout de suite interrogé sur la manière de créer un système interactif, à base de règles pouvant ressembler à celui-ci.

Après divers jours de réflexion nous avons donc décidé de mettre à profit nos experts dans le but de *créer un système expert à base de règle aidant au choix d'un sport selon différents critères*. Notre but premier étant de mettre en place une solution tout à fait interactive, nous permettant ainsi d'approfondir LISP d'une manière un peu nouvelle pour nous. En effet ayant appris différents langages avant celui-ci nous sommes familiarisés aux entrées/sorties. Cependant nous n'avons utilisé ces méthodes que très peu LISP et durant le semestre d'IA01 dans sa globalité. Notre sujet était donc tout trouvé, il ne nous restait plus qu'à la développer le plus intelligemment possible.

Voici la liste de nos experts nous ayons permis de prendre nos décisions futures :

Nicolas Goulhen : Etudiant UTCEEN, titulaire d'un diplôme d'état d'entraineur d'athlétisme, pratiquant au club Compiégnois, de niveau régional en saut à la perche,

Julien Marteau : Etudiant UTCEEN, ayant réalisé un SP02 en badminton à l'UTC,

Hubert Aulon : Entraineur et joueur de football de niveau 3ème division (national) à l'AFC Compiègne, seizième de finaliste de la coupe de France face à Lille en 2012 au stade Pierre Brisson de Beauvais,

Thomas Joly : Ancien étudiant UTCEEN désormais à Beauvais à l'IT2I, joueur de fédéral 2 au Rugby Club Compiégnois,

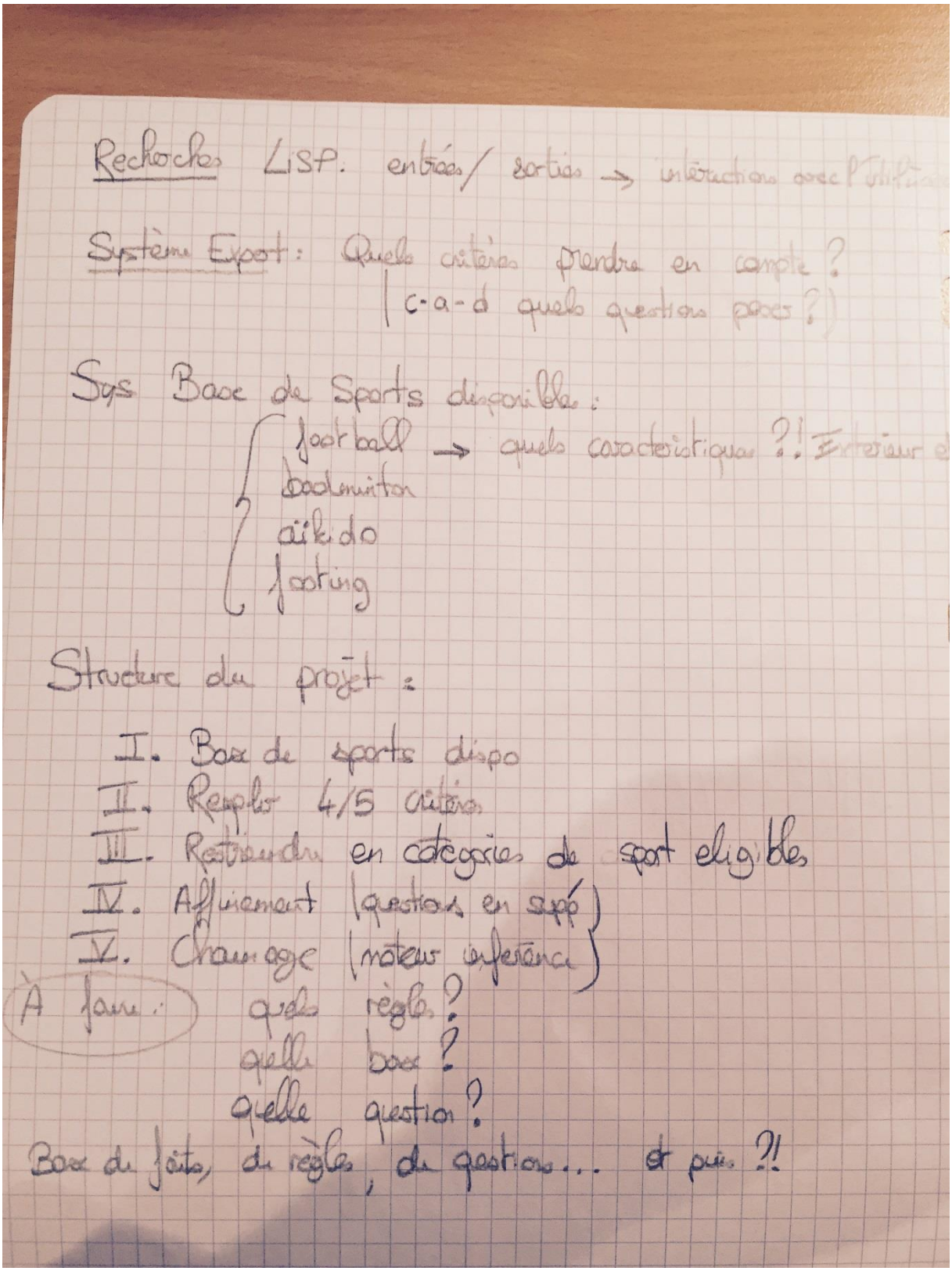
Marion Rbout : Championne de France junior de patinage artistique,

Nous avons mis à profit tous les sportifs de notre entourage, dans divers disciplines, à haut niveau ou non, à l'UTC ou pas.

En interviewant tour à tour chacun de ces intervenant et bien d'autres encore, en fréquentant régulièrement les sites spécialisés (notamment http://www. Sport365.com) ou la presse de connaisseurs (notamment une lecture assidue du journal Léquipe et un suivi attentif des actualités d'Eurosport), nous avons réussi à amassez de nombreuses connaissances.

Notre projet à suscité l'intérêt de nos collaborateurs qui ont fini par se prendre au jeu de l'intelligence artificielle, les plus curieux d'entre eux et Utcéen, regrettant que cette discipline ne soit pas ouverts aux étudiants du département GM.

Notre problématique décrite précisément et nos experts mobilisés nous nous sommes fixés un plan de route pour développer notre projet. En voici la feuille de route du 10 Décembre 2014 :



Henry s'occupait d'agrémenter nos différentes bases ainsi que les questions que nous allions poser pour restreindre notre base de sports disponible au départ. Nous voulions attendre au départ une base de 100 sports bien décrits.

De mon côté je m'attaquais à l'étude des entrée/sorties en LISP ainsi qu'à la réflexion des premières fonctions de services qui allait vite devenir obligatoire pour notre projet.

Plusieurs jours après et grâce à la participation active bien qu'éloignée de notre ami J.Marteau nous avons réussi à dégager quatre critère principaux quant à la qualification d'un sport (il nous fallut trancher entre plusieurs cependant la plupart de nos experts se sont accordés sur ceux-ci) :

- L'âge du participant, il nous semblait aberrant de proposer une partie de football américain à un enfant de 8 ans. Un de nos experts nous suggéra alors d'inclure une version spéciale de nos sports (une sorte d'initiation pour les sports à risque comme le trek en montagne ou autres, cependant, après discussions il nous est apparu que notre moteur d'inférence en deviendrait encore moins performant si nous multiplions les règles et les faits « possibles »). Nous avions déjà dénombrés 30 sports variés et intéressants, nous ne pouvions pas les considérer tous exhaustivement, il nous fallut faire un choix. Ce choix était aussi motivé par le fait que nous devions construire un système expert à base de règle à l'ordre 0+ Il nous fallait donc impérativement considérer d'autres opérateurs que celui de l'égalité. Ainsi un âge <= 15 ans (par exemple) pourrait être facilement utilisable dans notre moteur.

- Le second critère étant plus évident. Deux de nos experts habitant en ville et pratiquant régulièrement la musculation dans une salle de Gym à Compiègne il était évident de considérer les sports d'intérieurs et d'extérieurs. Nous pouvions de ce pas faire la différence entre quelqu'un recherchant une partie de football en hiver et quelqu'un en été. Il nous est alors apparu de remplacer cette règle d'intérieur/extérieur par un critère de saison. Cela aurait largement complexifié notre code. Nous gardions à l'esprit que nos idées devaient restées centrées sur l'idée principale du sujet : conduire un système expert d'ordre 0+ exploitable, utilisable et démontrable. Nous avons estimé, avec l'aide de nos experts qu'un critère binaire comme intérieur/extérieur reflétait plus directement une envie alors qu'un questionnement de saison, par les temps qui courent, n'aurait pas symbolisé grande chose. (se retrouver avec 15°C en hiver étant devenu assez courant nous ne pensions pas exploiter ce critère assez fidèlement)

- Les deux derniers critères que nous avons décidé de prendre en compte sont intimement liés. La distinction et le rappel de ces deux ci nous ont permis de nous faciliter ensuite. Quand nous posé la simple première question à nos experts : « Comment qualifieriez-vous votre sport » nous avons obtenus les réponses suivantes : « intense pour le badminton », « solo » pour la patineuse, « humain » pour le rugbyman et « épopée » pour le footballeur. Nous n'avions pas pensé à ce critère en premier lieu mais l'aide de nos experts nous a été plus qu'utile. «Solo »,

« Humain » et « Epopée » traduisaient tous les trois le même sentiment ou tout du moins la même idée : l'idée de groupe. Les quatre experts que nous avons choisis sont répartis également selon un critère : 2 sports individuels et 2 collectifs.

C'est alors que nous nous sommes quittés et où nous avons entrepris de développer notre projet, tout d'abord en algorithmique puis en LISP. Nous venions de dégrossir la première partie de notre sujet : l'expertise.

Remarque : Une quatrième question sera utilisée et rajoutée de dernière minute pour l'élaboration de notre système, l'explication de son intégration viendra en toute fin de rapport.

Après avoir une idée précise de notre développement en voici la structure finale :

Nous présenterons dans une première partie les fonctions associées à l'interaction home-machine et à la restriction des sports par questionnement de l'utilisateur. Dans une seconde partie les fonctions de services associées au chainage à proprement parler et le moteur d'inférence en lui-même.

Voici un schéma synthétisant les étapes de sollicitation de notre système expert.

Etape n°1 :

Nettoyage des variables globales

Etape n°2 :

Questions (première partie)

&

Initialisation de la liste des sports

Etape n°3 :

Regroupement en catégories + confrontation

&

Questions de sélection catégorie

Etape n°4 :

Lancement du moteur, analyse et retour solution

Les étapes 1 à 3 seront traitées dans la partie II. Tandis que la III occupera, avec ses fonctions de services la partie 4.

II. Interaction Homme-Machine et liste des sports

Remarque : Vous trouverez ci-joint à ce compte-rendu le fichier de code .cl commenté au maximum, ici ne seront rappelés que les grandes parties de chaque fonction et surtout comme elles s'articulent entre elles.

Aussi, le glossaire a été mis à jour avec les noms des variables locales et un résumé de leur utilité.

- (defun clean-var()
  (setq *Base-de-faits* '())

  (setq *Base-historique-question* '())

  (setq *liste-marques* '())

  (setq *liste-categories* '())
)

  Une fonction qui permet de vider le contenu des variables locales. On évite ainsi un conflit si on relance plusieurs fois le même programme ou bien si notre projet utilise des noms de variables déjà utilisées dans un traitement précédent. Ainsi on repart sur des bases saines.

- (defun verification_saisie (test)
  (let ((k nil))
  (loop
  (setq k (read))
  (if (not (member k (liste-arg test)))
  (format t "~%entree invalide, recommencez~%")
  (return k)
)
)
)
)

  Une première fonction qui vérifie que ce que l'utilisateur rentré au clavier est bien conforme à ce que l'on attend dans le programme. Ainsi on ne se retrouve pas avec des erreurs futures faisant buger le programme et nous obligeant à le redémarrer entièrement. Ce test immédiat fut d'abord incorporé à chaque saisi puis décentralisé dans une fonction bien particulière, ce qui nous a paru plus clair. La boucle est infinie tant que la saisie n'est pas conforme. Et le message d'erreur correspondant s'affiche. Cette fonction s'appuie sur liste-arg détaillée ci-après.

- ```
  (defun verification_saisie_liste (test)
   (let ((k nil))
    (loop
     (setq k (read))
     (if (not (member k test))
        (format t "~%entree invalide, recommencez~%")
      (return k)
      )
     )
    )
  )
  ```

  Cette fonction réalise la même tache que celle présentée ci-dessus sans s'appuyer sur la fonction liste-arg. En effet un seul choix est possible dans ce cas précis.

- ```
  (defun liste-arg(argument)
   (let ((liste))
    (dolist (y (caddr argument))
     (push (car y) liste)
     )

    liste
    )
  )
  ```

  Cette fonction est utilisée ci-dessus et permet de renvoyer une liste composée des éléments de l'argument. Cela a été notre seule solution pour faire fonctionner la fonction utilisant liste-arg ci-dessus. En effet sinon une erreur du member nous empêchait de réaliser correctement le test voulu.

- (defun Base-de-donnee-nom (item)
  (cadr (assoc 'nom item))
)
  Grace à celle-ci nous obtenons le nom d'un sport. Permet de parcourir la base de données de tous les sports en d'en récupérer les noms.

- (defun Base-de-donnee-valeur (sport cle)
  (cadr (assoc cle sport))
)

  Cette fonction « Base-de-donnee-valeur » renvoie la valeur associée à un sport ainsi qu'à une clé. Par exemple reverra individuel si on appelle cette fonction avec judo et type. Cela marche ainsi avec les sports de la base de données.

- (defun initialisation-marques ()
  (let (k)
  (dolist (k *Base-de-donnee*)
  (push (Base-de-donnee-nom k) *liste-marques*)
)
)
)

  Notre première étape consiste à marquer tous les sports de la base de données. On le fait grâce à la fonction ci-dessus qui parcourt la base de données, appelle la fonction «Base-de-donnée-nom » pour récupérer le nom de chaque puis push ce nom dans la listes des sports marqués appelée *liste-marque*

- (defun questions-preli()
  (let (reponse)
  (format t "Plutot sport d'{interieur} ou bien en {exterieur} ? ")
  (setq reponse (verification_saisie_liste '(interieur exterieur)))
  (marquage 'endroit reponse)

  (format t "Plutot sport {individuel} ou {collectif} ? ")
  (setq reponse (verification_saisie_liste '(individuel collectif)))
  (marquage 'type reponse)

  (format t "Quel est votre age ? {integer attendu} ")
  (setq reponse (read))
  (marquage 'age reponse)

  (format t "Pourquoi faites vous du sport ? {detendre social muscler depenser}, plusieurs choix possibles, a specifier entre parentheses ")
  (setq reponse (read))
  (marquage 'but reponse)
)
)

Cette liste de sports marqués ne correspondant pour l'instant à rien de concret si ce n'est la copie intégrale des sports que l'on a entré dans notre base de données. Il nous faut des critères filtrant. Les voilà avec ces questions préliminaires qui posent nos 3 questions vues précédemment ainsi que la dernière. Annoncée en lien avec la troisième c'est en effet le but de notre envie de faire du sport. Evidemment les motivations de chacun son propre cependant, la sollicitation du gérant de la salle de sport de l'UTC en partenariat avec MR. Vanicatte nous a permis de déterminer les 4 grandes classes de motivation pour faire du sport. Evidemment celles-ci sont arbitraires mais d'après l'expertise de ce monsieur, elles reflètent la réalité et finalement chacune de nos envies de sport peut se catégoriser dans l'un des quatre choix proposés : se détendre, rencontrer des gens, se muscler, ou se dépenser. Nous avons assimilé se vider la tête à « se dépenser » et retrouver la forme à « se muscler ».

- ```
  (defun marquage (cle val)
   (let (NouvelleMarque)
    (dolist (k *Base-de-donnee*)
     (when (member (Base-de-donnee-nom k) *liste-marques*)
      (case cle
       ('age (when (or (null (Base-de-donnee-valeur k cle)) (<= (Base-de-donnee-valeur k cle)
  val))
            (push (Base-de-donnee-nom k) NouvelleMarque)
            ))
       ('endroit (when (or (equal (Base-de-donnee-valeur k cle) 'deux) (equal (Base-de-
  donnee-valeur k cle) val))
              (push (Base-de-donnee-nom k) NouvelleMarque)
              ))
       ('but (dolist (y val)
            (when (member y (Base-de-donnee-valeur k cle))
             (if (not (member (Base-de-donnee-nom k) NouvelleMarque))
              (push (Base-de-donnee-nom k) NouvelleMarque)


              ))))

       (otherwise (when (equal (Base-de-donnee-valeur k cle) val)
              (push (Base-de-donnee-nom k) NouvelleMarque)
              ))
       )
      )
     )
    (setq *liste-marques* NouvelleMarque)
    )
  )
  ```

  Il est nécessaire désormais de restreindre la liste des sports avec les critère obtenus par réponse aux 4 questions posées à l'utilisateur. Ainsi après ce filtrage réalisé par la fonction « marquage » et la variable intermédiaire « NouvelleMarque », on obtient dans la liste des sports marqués *list-marques* seulement ceux qui coïncident avec les réponses de l'utilisateur.
  ```

- ```
  (defun liste-categorie( liste-des-sports)
   (let (categories)
    (dolist (k *Base-de-donnee* categories)
     (when (member (Base-de-donnee-nom k) liste-des-sports)

       (when (not (member (Base-de-donnee-valeur k 'categorie) categories))
        (push (Base-de-donnee-valeur k 'categorie) categories)
        )
       )
     )
    )
   )
  ```

  On passe ensuite en catégories. Après avoir récupéré la liste des sports filtrés on « construit » une liste des catégories résumant l'ensemble des sports filtrés. Un dolist accompagné d'un push sur une variable locale catégorie nous permet de construire cette liste. On se base sur le paramètre qui nous intéresse : la liste des sports qui sera passée en paramètre. Ainsi on assimile chaque sport de cette liste à sa correspondance dans la base de données pour retrouver sa catégorie et ainsi la stocker. On a seulement stocké le nom du sport dans la liste des sports.

- ```
  (defun Questions-spec-categorie (liste-des-sports)
   (let ((reponse nil)(list-categories (liste-categorie liste-des-sports)))
    (if (member 'natation list-categories)
       (progn
        (format t "L'eau vous derange-t-elle ? ")
        (if (eq (verification_saisie_liste '(oui non)) 'oui)
           (push 'eau reponse)
         ))
     )
    (if (member 'athletisme list-categories)
       (progn
        (format t "Etes vous quelqu'un d'endurant ? ")
       (if (eq (verification_saisie_liste '(oui non)) 'oui)
           (progn
            (push 'endurance reponse)
            (push 'athletisme reponse)
            (push 'extreme reponse)
            )
         ))
      (if (member 'endurance list-categories)
       (progn
        (format t "Etes vous quelqu'un d'endurant ? ")
        (if (eq (verification_saisie_liste '(oui non)) 'oui)
  ```

```
            (progn
              (push 'endurance reponse)
              (push 'athletisme reponse)
              (push 'extreme reponse)
              )
            ))
      (if (member 'extreme list-categories)
        (progn
          (format t "Etes vous quelqu'un d'endurant ? ")
          (if (eq (verification_saisie_liste '(oui non)) 'oui)
            (progn
              (push 'endurance reponse)
              (push 'athletisme reponse)
              (push 'extreme reponse)

              )
            ))
        )
      )
    )

(if (member 'combat list-categories)
    (progn
      (format t "Etes vous allergique aux sports de contact ? ")
      (if (eq (verification_saisie_liste '(oui non)) 'oui)
                                              ;;(push 'contact reponse)
          (push 'combat reponse)
          ))
    )

(if (member 'fitness list-categories)
    (progn
      (format t "Etes vous deranges par les sports rythmés ? ")
      (if (eq (verification_saisie_liste '(oui non)) 'oui)
          (push 'fitness reponse)
          ))
    )
(if (member 'glisse list-categories)
    (progn
      (format t "Appreciez-vous la vitesse ? ")
      (if (eq (verification_saisie_liste '(oui non)) 'oui)
          (push 'glisse reponse)
        (push 'extreme reponse)
        ))
```

```
 (if (member 'extreme list-categories)
    (progn
     (format t "Aimez-vous la vitesse ? ")
     (if (eq (verification_saisie_liste '(oui non)) 'oui)
        (push 'extreme reponse)
      (push 'glisse reponse)
      ))
  ))
(if (member 'balle list-categories)
   (progn
    (format t "Aimez-vous les sports de ballon ? " )
    (if (eq (verification_saisie_liste '(oui non)) 'oui)
       (push 'balle reponse)
     ))
 )
(if (member 'endurance list-categories)
   (progn
    (format t "Avez-vous un mental d'acier ? " )
    (if (eq (verification_saisie_liste '(oui non)) 'oui)
       (push 'endurance reponse)
     ))
 )
(if (member 'equitation list-categories)
   (progn
    (format t "Savez vous monter a cheval ? " )
    (if (eq (verification_saisie_liste '(oui non)) 'oui)
       (push 'equitation reponse)
     ))
 )
(REMOVE-DUPLICATES reponse)))
```

Après plusieurs essais dans l'état actuel des choses on se rend vite compte que notre base de sports marqués filtrés est rapidement saturée et inutilisable. Ainsi après les questions préliminaires on a, en accord avec nos experts, tenté de spécifier encore chaque catégorie de sport. En effet certains sports proches présentaient les même caractéristiques selon notre modèle (même type, même âge limite, même lieu de pratique, même envie de pratiquer). Ainsi il fallait, pour chaque catégorie spécifié encore un peu plus les envies de l'utilisateur pour restreindre la possibilités et donc les sports marqués. Cette longue fonction utilise simplement la liste des catégories de sports filtrés, et propose, en fonction des catégories présentes, des questions spécifiques pour construire « reponse », une liste restreinte encore plus. Cette liste sera sans doublons grâce à la fonction de LISP qui nous a été conseillée en TD par MR. Lenne : « remove-duplicates »

- (defun sports-valide (sport categories)

```
 (let (liste-des-sports S)
   (dolist (k sport)
     (dolist (y *Base-de-donnee*)
       (when (eq k (Base-de-donnee-nom y))
         (setq S y)
         (return nil)
         )
       )
     (when (member (Base-de-donnee-valeur S 'categorie) categories)
       (push k liste-des-sports)
       )
     )
   liste-des-sports
   )
)
```

On réalise ensuite l'intersection de la liste des sports étendue avec la liste des catégories épurée grâce à la fonction « sports-valide » ci-dessus. Ainsi on renvoie une liste des sports correspondant fidèlement aux réponses de l'utilisateur. Jusqu'alors on n'a effectué aucun traitement, simplement de la manipulation de données de l'utilisateur en prévision d'un traitement.

- ```
  (defun choix ( categories )
   (let ((choix))
    (if (> (list-length categories) 1)
       (progn
        (format t "Choissisez une catégorie de sport parmi les suivantes :")
        (print categories)
        (setq choix (verification_saisie_liste categories))
        (setq *liste-categories* choix)
        )

      (progn
       (setq *liste-categories* (car categories))
       )
      )
    )
  )
  ```

  Le dernier choix de l'utilisateur intervient ici, on lui demande de choisir parmi la catégorie éligible selon le traitement précédent. Bien sûr, on ne lui laisse pas le choix si ses précédentes réponses ont abouti à l'élection d'une seule et unique catégorie possible.
  ```

Nous avons ainsi toutes les fonctions de manipulations de données nécessaires à l'écriture de la fonction globale de lancement du processus de dialogue puis de traitement des données recueillies, voici comment nous ordonnons les étapes précédemment détaillées :

```
(defun lancement-proc ()
  (let (liste-bis listecategorie sport)
    (clean-var) ;; nettoie les variables
    (initialisation-marques) ;; marque tous les sports
    (questions-preli) ;; On obtient une liste de sports suite aux questions
    (setq listecategorie (choix-categorie *liste-marques*)) ;; Pour en déduire une liste de catégories
    (setq liste-bis (sports-valide *liste-marques* listecategorie)) ;; On obtient une liste de sports épurée de certaines catégories
    (choix listecategorie) ;; Puis on charge la catégorie finalement élue dans *liste-categories*
    (setq *Base-de-faits* `(,*liste-categories*));; On charge dans *Base-de-faits* la catégorie élue

    (when (member 'eau listecategorie)
      (push 'oui-aquatique *Base-de-faits*)          ;; Cas particulier traité ici des sports d'eau
      )

    (setq *liste-marques* (sports-valide liste-bis *Base-de-faits*)) ;; Liste restreinte et finale des sports marqués
    (if (setq sport (moteur_inference));;: On lance le moteur d'inférence pour analyser nos bases et les confronter
        (format t "Le sport qui vous est le plus adapté est: ~A" sport)
      (echec-sport)
      )
    )
)
```

Il ne nous reste plus qu'à détailler le rôle et les fonctions de services spécifiques au moteur d'inférence pour compléter le détail de notre analyse.

III. Moteur d'inférence et fonctions de service

- (defun existe (cle)
  (member cle *Base-de-faits*)

)

On vérifie ici tout d'abord si une clé appartient à la base de fait

- (defun test-regle-candidate ()
  (let (candidats premisses but Prem)
   (dolist (k *Base-de-regle* candidats)
    (setq premisses (cadr k))
    (setq but (caddr k))

    (setq Prem (analyse-premisse premisses but))
    (when Prem
     (dolist (k Prem)
      (push k candidats)
)
)
)
)
)

  Cette fonction spéciale boucle sur une liste de règles. Récupère les prémisses de chaque règle. Passe par la fonction d'analyse des prémisses pour savoir si cette règle est candidate ou non, si oui on la push dans la variable locale candidate

- ```
  (defun analyse-premisse (premisses but)
   (let (baseExist result)
    (if (existe but)
       nil
     (dolist (k premisses)
      (if (member k *Base-de-faits*)
         (setq baseExist T)
       (push k result)
       )
      )
     )
    (if baseExist result nil)
    )
   )
  ```

  La fonction ci précédemment a besoin de savoir si les prémisses d'une règle sont dans la base afin de déterminer si cette règle est candidate ou non. Pour cela on utilise la fonction « analyse-premisse » qui teste si les prémisses sont dans la *Base-de-faits* ou non et retourne celles qui en font partie. Ainsi elle prend en paramètre premisses qui est une liste et but, résultat de l'implication symbolisant l'application d'une règle. Si le but existe déjà on renvoie nil, inutile de l'ajout de nouveau (un pushnew aurait pu être aussi utilisé). Sinon on boucle sur les prémisses vérifiant si elles sont toutes vérifiées pour les retourner (ou non).
  ```

- ```
  (defun recup-questions (missing base-q)
   (let (result)
    (loop
     (dolist (k (caddr (car base-q)))                    ;; on parcourt le 3e argument d'une question
      (when (not (member (car (car base-q)) *Base-historique-question*))
       (when (equal (cadr k) (car missing)) (setq result (car base-q)) (return nil))
       )
      )
     (setq bq (cdr base-q))
     (when (null base-q) (return nil))
     )
    result
          )
  )
  ```

  Cette fonction génère la liste des questions en fonction du mot clé, en testant si ces questions n'ont pas déjà été posées. Prenant en paramètre le mot clé à tester et la liste de questions. On teste si la question n'est pas déjà dans l'historique, sinon on l'ajoute à result, variable local contenant la question et on la retournera à la fin du traitement de cette fonction

- ```
  (defun verif-regle ()
   (let (premisses but baseExist)
    (dolist (k *Base-de-regle*)
     (setq premisses (cadr k))
     (setq but (caddr k))
     (setq baseExist t)

     (when (not (member but *Base-de-faits*))
      (dolist (y premisses)
       (when (not (member y *Base-de-faits*)) (setq baseExist nil))
       )
      (when baseExist (push (car but) *Base-de-faits*))
      )
     )
    )
  )
  ```

  Il nous faut ensuite avancer dans le chainage ; on boucle sur la base de règle en analysant la base de faits. Si des premières d'une règle sont toutes présentes on ajoute le but à la base de faits. Ainsi le chainage s'effectue par lui-même grâce à « verif-regle ».

- (defun chainage-avant (missing)
  (let ((base-q (cdr (assoc *liste-categories* *Base-de-questions*))) item reponse baseExist baseTrouvee)
  (loop
  (setq item (recup-questions missing base-q))
  (when item
  (loop
  (format t (cadr item))
  (setq reponse (verification_saisie item))

  (dolist (k (caddr item))
  (when (equal reponse (car k)) (push (cadr k) *Base-de-faits*) (verif-regle) (setq baseExist t))

)
  (when baseExist (return nil))
)
  (push (car item) *Base-historique-question*)
  (dolist (k (caddr item))
  (when (member (car k) missing) (delete (car k) missing))
)
  (setq baseTrouvee t)
  (return t)
)
  (setq missing (cdr missing))
  (when (null missing) (return nil))
)
  baseTrouvee
)
)

  Cette fonctionne st la fonction centrale de notre moteur d'inférence. En effet elle traille sur la liste des règles candidates. Passe par recup-question pour avoir la liste des questions correspondant à la catégorie choisie, pose la question , récupère la réponse à celle-ci, tester la valeur grâce aux fonctions de test d'entrée, ajoute la réponse à la base de faits (enrichissement) puis avance dans le chainage grâce à la fonction vue ci-dessus. Elle est le pivot central de notre moteur.

- ```
  (defun moteur_inference ()
   (let (baseTrouvee sport)
    (setq baseTrouvee (chainage-avant (test-regle-candidate)))
    (setq sport (sport-final))
    (when baseTrouvee
     (when (not sport) (setq sport (moteur_inference)))
     )
    (if sport
       sport
     nil
     )
    )
  )
  ```

  Notre moteur est alors articulé ainsi et fait le lien entre les deux parties que l'on vient d'aborder (II. Et III.).

- ```
  (defun sport-final ()
   (let (sport)
    (dolist (k *Base-de-faits*)
     (when (member k *liste-marques*) (setq sport k) (return nil))
     )
    sport
    )
  )
  ```

  Cette fonction teste au final s'il existe un élu par correspondance des deux parties traitée par le moteur d'inférence. Ainsi nil est renvoyé par cette fonction si aucun sport marqué ne coïncide.

- ```
  (defun echec-sport()
   (let ((reponse)(sport))
    (dolist (k *Base-de-donnee*)
     (if (member (cadr(assoc 'nom k )) *Base-de-faits*)
                                        (push (cadr(assoc 'nom k )) sport)
      )
     )
    (if sport
      (format t "Aucun sport ne correspondant exactement à toutes  vos réponses n'a été
  trouvé, le sport le plus proche est le suivant : ~A" (car sport) )
     (progn
      (format t "Aucun sport n'a été trouvé ! Souhaitez-vous recommencer ? {oui/non} ")
      (setq reponse (verification_saisie_liste '(oui non)))
      (if (eq reponse 'oui)
         (lancement-proc)
       )
      )

     )
    )
  )
  ```

  Afin de coller encore plus à la réalité et à nos envies nous avons essayé de programmer cette fonction qui donne le sport le plus proche si aucun ne coïncide complétement avec les choix de l'utilisateur. Nous avons décidé de proposer un lancement du programme une nouvelle fois ci aucun sport ne coïncide et qu'aucun n'était même au plus proche. (On relance ainsi notre fonction globale : lancement-proc)
  ```

IV. Limites et ambitions futures de développement :

Bien que notre projet nous satisfasse dans ses grandes largeurs nous nourrissons quelques regrets quant au développement de celui-ci :

- Une base de sports peu étoffée : même si nous avons essayé de varier les sports, les domaines, les caractéristiques et les origines nous aurions pu faire une liste presque exhaustives des sports pratiqués en France tout du moins (et en Chine avec l'aide d'Henry).

- Seulement quatre critères pris en compte et discutables : même si nous deux derniers critères se rejoignent, ils ne sont donc pas indépendants totalement. En effet nous ne choisissons pas un sport collectif si nous ne voulons pas rencontrer des gens. Notre système présente donc une sorte d'incohérence qui nous dérange. Deux critères de sélection de notre base de sports de départs ne sont pas incompatibles. Même si cela ne nous a pas empêché de réaliser notre projet initial nous aimerions continuer à développer ce projet en ajoutant plus de critères initiaux et bien sur plus de questions subsidiaires (les questions de secondes parties servant à la restriction de la catégorie). En effet l'étude de MyCin nous a montré à quel point les solutions envisagées au départ du développement permettent ensuite de couvrir le plus large panel d'éventualités et d'ainsi éviter les incohérences. En effet avec plus de critères de décisions nous arrivons à moins d'incohérence. En effet en essayant d'anticiper les résultats du moteur certains sports sont oubliés ou d'autres sont intégrés tout à fait logiquement d'après notre model mais nettement moins selon la réalité. C'est ici que nous nous heurtons à un problème fondamental : nous sommes bloqués par nos choix *a priori* et nos système n'apprends pas de lui-même.

- Notre ambition trop grande à vouloir faire un système interactif nous a détourné un petit peu du but initial du projet : de coder entièrement un système expert capable de raisonner par chainage avant et arrière. Nous avons voulu sans doute trop nous appuyer sur MyCin existant avec ses interactions multiples.

- Notre projet bien que complexe nous apparait difficilement accessible et compréhensible au premier abord. De nombreuses listes/variables globales/… sont utilisées. Rendant difficile la compréhension primaire de notre ambition.

- Trop de fonctions (de services ou non) sont codées en dur, ce qui nous empêche une évolution optimale et une adaptation idéale (l'auto-renforcement de notre modèle aurait pu et dû être envisagé, un apprentissage des cas passés et un traitement plus rapide des prémisses également).

- Les symboles choisis sont discutables et ne permettent pas de se repérer dans notre programme

<u>MEMO</u> : Pour résoudre ce problème nous avons ajouté en dernière minute un glossaire des symboles utilisés et leur signification

De même les noms des fonctions ont été améliorés en toute fin de développement

- Nous aurions pu proposer une interface de modification de la base de règles : cette flexibilité pourrait améliorer la portabilité de notre programme et ainsi l'adapter à d'autres cultures, d'autres pays.

- Il a été évoqué en stade de développement d'enregistrer les parcours dans un fichier afin de pouvoir retracer celui-ci et ainsi l'améliorer plus facilement. Nous avons affiché le chemin parcouru à l'aide de nombreux « print » lors du développement. Nous devons les avoir tous supprimés mais nous nous excusons par avance si vous en rencontrez un qui n'affiche rien.

- Nous aurions aimé pouvoir intégrer la gestion des accents dans l'interpréteur LISP pour éviter certaines coquilles d'affichage tout à fait désagréables et disgracieuses. Cependant ayant rencontré des problèmes dans notre chainage de dernière minute nous n'avons pas poussé les investigations plus loin.

- La gestion de notre cas « Erreur » aurait pu être améliorée : en effet nous avons essayé de même en place un système probabiliste de correspondance d'une solution comme nous l'avons envisagé dans le TP sur la voiture d'occasion. En effet, nous n'avons pas testé toutes les possibilités de réponse aux différentes questions de restriction de la base de sports et certaines combinaisons de réponse amènent à un résultat « vierge ». Cependant notre système a été pensé, au départ, pour toujours donner une réponse ou du moins « suggérer » quelque chose à son utilisateur. Pourquoi ne pas classer les résultats en classe en fonction de leur degré de concordance avec les critères rentrés par l'utilisateur. De fixer une pondération aux critères puis une valeur seuil de décision pour ensuite choisir le « cas le plus proche ».

- Le système probabiliste envisagé aurait pu être poussé au maximum en fournissant à chaque prémisse et à chaque conclusion un nombre réel compris entre 0 et 1. Celui-ci qualifiant la certitude de l'implication. On pourrait alors raisonner en réseau et successivement attribuer une valeur de certitude aux relations et aux conclusions. Nous aurions ainsi un nouveau classement de parcours en profondeur et en largeur en fonction de la certitude de ceux-ci. En effet, le chemin du plus court n'est pas toujours le meilleur si le prix à payer est excessif. Le prix pouvant facilement être assimilable au produit des probabilités des prémisses par les conclusions des implications (une moyenne arithmétique simple sur les probabilités des prémisses pouvant être envisagée au cas où elles sont nombreuses).

Conclusion :

Nous pouvons affirmer sans crainte que la réalisation de ce projet restera sans doute une très bonne expérience de programmation informatique. La manière de penser spécifique à l'intelligence artificielle et ses sans cesse remises en cause : que ce soit sur le fond des données représentées pour leur exploitation future ou la forme de celles-ci grâce à tous les outils disponibles de représentation des connaissances lors du développement. Bien que nous soyons seulement à quelques balbutiements primaires en développement LISP et que nos considérations de représentation des connaissances restent tout à fait primaire nous avons apprécié nous poser ces diverses questions. Ce TP n°3 restera le premier projet d'envergure en IA et en GI en général pour un des deux membres de ce groupe. Et bien que tout ne soit pas satisfaisant, l'envie de l'optimiser et d'y ajouter de petites touches personnelles pendant l'inter-semestre qui arrive reste un point positif. Nous avons cru, et cela est dit positivement, parfois retourner en cours de philosophie. Cette philosophie que nous fait nous questionner sur le monde qui nous entoure. Ce projet restera très formateur du point de vu strict de notre formation d'ingénieur. LISP étant un langage assez facilement assimilable nous avons pu nous concentrer sur le fond de notre raisonnement. Malheureusement nous n'avons pu consulter que deux projets similaires à ce TP numéro 3 d'IA donc notre expérience ne nous permet pas de porter un regard plus critique que celui qui a été le nôtre durant la rédaction de ce rapport. Cependant ce projet nous a permis de faire découvrir l'intelligence artificielle à notre entourage et notamment à nos volontaires experts, un domaine qui reste source de questionnement et de fascination pour beaucoup.

Annexe :

Glossaire :

A l'aide de définitions de variables globales nous utilisons tout au long de ce TP les variables suivantes :

Base-de-donnee : contenant tous les sports que nous avons sélectionné

Base-de-faits : base construite au fur et à mesure

Base-de-questions : contenant les questions spécifiques à chaque catégorie de sport pour spécifier

Base-historique-question : les questions déjà posées au préalable

Base-de-regle : les règles que nous avons mis en place avec nos experts avec un identifiant rappelant la catégorie du sport en question

liste-marques : liste des sports marqués (tous au départ)

liste-categories : liste des catégories issue des sports sélectionnés