Rapport TP3

Système expert d'ordre 0+

Jarvis Solution

ALLIANCE SORBONNE UNIVERSITÉ utc

Sommaire

Introduction

Nous avons été amenés à travailler durant le dernier TP de IA01 sur un système expert. Ce dernier TP est une mise en pratique des notions qui y sont liées. Pour cela, le sujet est assez large et nous laisse libre de le réaliser sur le sujet de notre choix. Nous étions d'abord partis sur un système expert permettant de choisir une boisson en fonction d'un état donné, mais ce sujet ne couvrait pas toutes les notions d'un système expert. C'est pourquoi nous avons choisi ce sujet : *Jarvis solution*.

Nous avons décidé de réaliser un système expert nommé *Jarvis solution*, qui a pour objectif de trouver une solution adaptée aux problèmes informatiques rencontrés par l'utilisateur. Pour cela, nous avons construit une base de règles reposant sur un arbre (dont vous pouvez trouver un schéma en annexe de ce rapport).

Notre système expert comprend deux moteurs d'inférences : le chainage avant, qui permet de trouver la solution la plus adaptée, et le chainage arrière, qui vérifie si une solution correspond bien au problème posé.

Nous avons mis en place différentes fonctions et jeux de données de test pour vérifier le bon fonctionnement de notre système. Tous les codes et commentaires explicatifs sont présents dans le fichier jarvis.cl

1 Choix du sujet du système expert

1.1 Premières idées de système expert

Notre première idée de sujet pour ce TP était de créer un système expert permettant de choisir un plat en fonction d'un état donné (par exemple, si l'utilisateur se sent fatigué, stressé, etc.). Nous avons rapidement constaté que ce sujet était trop commun et pouvait facilement dériver vers le choix de plats en fonction d'aliments, marmiton par exemple, ce que nous ne souhaitions pas.

Notre seconde idée de sujet était similaire à la première, mais portait sur le choix d'une boisson en fonction d'un état donné. Nous avons réalisé l'arbre qui serait utilisé pour cette idée, mais nous avons constaté qu'il y avait trop de critères récurrents. Par exemple, les critères de goût, de température, étaient présents à de nombreuses reprises. Le sujet risquait de devenir trop complexe à mettre en place et ne répondait pas entièrement aux attentes d'un système expert.

1.2 Idée finale : Jarvis solution

Après avoir abandonné nos deux premières idées de sujets, nous avons cherché un thème plus adapté à nos objectifs pour ce TP sur les systèmes expert. C'est ainsi que nous avons opté pour le sujet que nous avons intitulé *Jarvis solution*.

Le nom "Jarvis" nous est venu en tête, car nous sommes de grands fans de Tony Stark et de l'univers Marvel. Nous avons donc choisi de reprendre le nom du système de gestion de l'armure de Iron Man, qui était capable de résoudre de nombreux problèmes technologiques.

En ce qui concerne le contenu du sujet, nous avons choisi de mettre en place un système expert capable de trouver une solution adaptée aux problèmes informatiques rencontrés par l'utilisateur. En tant que passionnés d'informatique, nous avons été confrontés à de nombreux problèmes de ce genre et nous avons de ce fait estimé que ce sujet était pertinent et utile.

2 Choix des moteurs d'inférences

Pour mettre en place notre système expert "Jarvis solution", nous avons choisi de travailler avec deux moteurs d'inférences : le chainage avant et le chainage arrière. Le chainage avant va nous permettre de trouver une solution à un problème décrit par l'utilisateur. Le chainage arrière va quant à lui permettre de vérifier qu'une solution est appropriée pour résoudre un problème décrit par l'utilisateur.

2.1 Chainage avant

Le chainage avant est un moteur d'inférence qui permet de trouver la solution la plus adaptée au problème posé. Nous avons choisi d'utiliser ce moteur d'inférence, car c'est celui qui nous a semblé être le plus adapté à notre système expert.

Pour mettre en place le chainage avant, nous avons procédé de la manière suivante :

1. Nous avons commencé par demander les problèmes les plus courants par ordre de probabilité (c'est-à-dire, les problèmes les plus fréquemment rencontrés).
2. Si le problème posé n'était pas un problème courant, nous avons alors cherché la solution adaptée en posant une série de questions qui étaient adaptées en fonction du noeud sur lequel nous nous trouvions dans l'arbre. Cela nous a permis d'éviter de suivre des règles qui ne nous étaient pas utiles pour l'étape suivante.

Nous avons testé le chainage avant sur plusieurs exemples de problèmes que nous saisissions et avons constatés qu'il était efficace pour trouver des solutions adaptées aux problèmes posés.

2.2 Chainage arrière

Le chainage arrière est un moteur d'inférence qui permet de vérifier si une solution donnée correspond bien au problème posé. Nous avons choisi d'utiliser ce moteur d'inférence car il nous a semblé complémentaire au chainage avant et nous a permis de vérifier l'efficacité de notre système expert "Jarvis solution".

Pour mettre en place le chainage arrière, nous avons procédé de la manière suivante :

1. Jarvis demande à l'utilisateur la solution qu'il souhaite tester.
2. Ensuite, le système expert interroge la base de règles pour vérifier si cette solution était adaptée au problème posé. Pour ce faire, ce dernier l'arbre de déduction en partant de la solution et en remontant jusqu'à l'état initial.
3. Si la solution n'était plus atteignable depuis un état, cela signifiait que la solution ne correspondait pas au problème. Le système expert met donc fin à la recherche et indique les résultats à l'utilisateur.

Nous avons testé le chainage arrière sur plusieurs jeux de données et avons constaté qu'il était efficace pour vérifier l'adaptation de la solution aux problèmes posés. Nous avons également effectué des tests de performance pour évaluer la rapidité du chainage arrière et avons constaté que son temps d'exécution était raisonnable.

3 Explication de nos chainages

3.1 Explication du chainage avant

Avant d'appeler la fonction de chainage avant dans le menu principal, nous proposons à l'utilisateur de compléter, s'il le souhaite, la liste des problèmes les plus courants. En effet, cette liste peut évoluer avec le temps et les technologies, et notre système expert permet ainsi une certaine modularité dans son expertise de problèmes. L'utilisateur peut choisir de ne pas compléter cette liste et elle restera celle que nous avons définie sur la base des liens présents dans les sources. Voici les codes des fonctions utilisées pour l'ajout de problème courant :

```
(defun demander-nom ()
  (let ((nom nil))
    (loop
      (format t "~&Entrez le nom de la solution parmis cette liste:
")
      (mapcar #'print (mapcar #'first *solutions*))
      (format t "~&Votre saisie :")
      (setq nom (read))
      (if (member nom (mapcar #'first *solutions*))
        (return nom)
        (format t "~&Ce nom de probleme n'est pas valide. Veuillez
reessayer.~%")))))
```

FIGURE 1 – demander-nom

```
(defun remove-elements-with-zero-cadr (lst)
  (let ((result nil))
    (dolist (elt lst result)
      (if (not (= 0 (cadr elt)))
        (push elt result)))))
```

FIGURE 2 – remove-elements-with-zero-cadr

```
(defun ajouter-pb-courant ()
  (let ((nom nil) (description nil) (somme_proba 0) (prob_temp 0))
    (setq nom (demander-nom))
    (format t "~&Entrez la description du probleme : ")
    (setq description (concatenate 'string "~&" (read-line)))
    (pushnew (list nom 1 description) *pb_courant*)
    (loop while (not (= (apply #'+ (mapcar #'second *pb_courant*))
1)) do
      (format t "~&La somme des probabilites doit etre egale a 1.~&
Veuillez verifier les probabilites des problemes deja enregistres
et entrer de nouvelles probabilites si necessaire.")
      (setq somme_proba 0)
      (dolist (pb *pb_courant*)
        (format t "~&Somme proba actuelle : ~,3f~&Reste : ~,3f"
somme_proba (- 1 somme_proba))
        (format t "~&Entrez la probabilite du probleme '~a' (entre
0 et 1) : " (first pb))
        (setq prob_temp (read))
        (setf (second pb) prob_temp)
        (setq somme_proba (+ somme_proba prob_temp))
      )
    )
    (setq *pb_courant* (remove-elements-with-zero-cadr *pb_courant
*))))
```

FIGURE 3 – ajouter-pb-courant

Notre moteur de résolution de problèmes utilise une technique de chainage avant afin de permettre à l'utilisateur de trouver une solution adaptée à son problème. Ce mécanisme fonctionne de la manière suivante :

1. Interrogation de l'utilisateur : Le moteur commence par poser des questions à l'utilisateur dans le but de mieux comprendre le problème rencontré. Ces questions visent à identifier les symptômes et à éliminer certaines hypothèses.
2. Reconnaissance du problème : Si l'utilisateur reconnaît son problème parmi les propositions du moteur, celui-ci lui propose alors une solution adaptée et s'arrête.
3. Expertise du problème : Dans le cas où l'utilisateur ne reconnaît pas son problème, le moteur continue à poser des questions afin de mener une expertise plus approfondie. Il s'appuie sur un ensemble de règles préétablies pour éliminer les hypothèses de manière systématique et trouver la solution la plus adaptée.

Le moteur de chainage avant est donc un outil efficace pour résoudre des problèmes en posant des questions et en utilisant une démarche itérative basée sur des règles préétablies.

La fonction de chainage avant repose sur les fonctions suivantes :

1. get-description-val-regles prend en argument une règle et renvoie la valeur de la description de cette règle.
2. get-nom-var-regles prend en argument une règle et renvoie le nom de la seconde variable associée à cette règle.

3. find-next-regles prend en argument une description et renvoie la liste des règles qui ont cette description.
4. sort-list-of-lists prend en argument une liste de listes et les trie par ordre décroissant du second élément de chaque liste, ici le second élément est la probabilité d'un problème

Voici les codes des différentes fonctions utilisées dans le processus de résolution de problèmes par chainage avant :

```
(defun get-description-val-regles (regle)
  (let ((val nil))
    (dolist (condition (conditions_regle regle))
      (if (member 'Description condition)
        (setq val (caddr condition))
      ))
    val))
```

FIGURE 4 – get-description-val-regles

```
(defun get-nom-var-regles (regle)
  (cadr (cadr (conditions_regle regle))))
```

FIGURE 5 – get-nom-var-regles

```
(defun find-next-regles (desc)
  (let ((next-regle nil))
    (dolist (regle *regles*)
      (if (eq (get-description-val-regles regle) desc)
        (pushnew regle next-regle)
      ))
  next-regle))
```

FIGURE 6 – find-next-regles

```
(defun sort-list-of-lists (ma-liste)
  (if (> (length ma-liste) 1)
      (sort ma-liste (lambda (a b) (> (second a) (second b))))
    ma-liste
    ))
```

FIGURE 7 – sort-list-of-lists

Voici le code correspondant au mécanisme de chainage avant décrit précédemment :

```
(defun chainage-avant ()
  (let ((val nil) (next-regles nil) (var nil) (ok t) (
rep_pb_courant nil) (pb_courant *pb_courant*))
    (setq pb_courant (sort-list-of-lists pb_courant))
    (format t "~&Commencons par voir les problemes les plus
courants.")
    (dolist (pb_courant pb_courant)
      (setq rep_pb_courant nil)
      (format t "~&Votre probleme correspond a la description
suivante :")
      (format t (caddr pb_courant))
      (loop while (not (OR (eq rep_pb_courant 'y) (eq
rep_pb_courant 'n))) do
        (format t "(Y/N)~&Votre choix : ")
        (setq rep_pb_courant (READ))
      )
      (if (eq rep_pb_courant 'y)
          (progn
            (ajouter_fait 'solution (car pb_courant))
            (afficher_solution (cadr (assoc 'solution *faits*)))
            (return-from chainage-avant T)
          )))
    (format t "~&Votre probleme ne semble pas etre un probleme
courant.~&Decrivez moi plus en details votre probleme.")
    (setq val (demander_valeur_question 'Type))
    (if val
        (progn
        (ajouter_fait 'Type val)
        (ajouter_fait 'Description val)
        (loop while (not (cadr (assoc 'solution *faits*))) do
              (setq next-regles (find-next-regles (cadr (assoc '
Description *faits*))))
              (setq var (get-nom-var-regles (car next-regles)))
              (setq val (demander_valeur_question var))
              (ajouter_fait var val)
              (loop for regles in next-regles do
                (progn
                  (setq ok t)
                  (dolist (condition (conditions_regle regles))
                    (setq ok (appartient_aux_faits condition))
                  )
                  (if ok
                    (ajouter_fait (cadr (conclusion_regle regles))
(caddr (conclusion_regle regles)))

                  ))))
        (afficher_solution (cadr (assoc 'solution *faits*)))
      ))))
```

FIGURE 8 – chainage-avant

3.2 Explication du chainage arrière

Étant donné que l'algorithme du chainage arrière utilisé est récursif, deux fonctions ont été nécessaires pour implémenter ce moteur d'inférence : une pour faire choisir à l'utilisateur une solution à tester, et une qui va faire le vrai travail de chainage arrière.

Voici la première :

```
(defun chainage-arriere ()
  (let ((sol nil))
    (format t "~2&Quelle solution vous semble appropriee pour le
probleme que vous avez ?~&")
    (loop for solution in *solutions* do
      (progn
        (format t "~2&~a : " (car solution))
        (format t (cadr solution))))
    (loop while (not (assoc sol *solutions*)) do
      (progn
        (format t "~2&Votre choix : ")
        (setq sol (read))))
    (if (chainage_arriere `(eq Solution ,sol))
      (format t "~&La solution ~a est la bonne" sol)
      (format t "~&La solution ~a n'est pas la bonne" sol))))
```

FIGURE 9 – chainage-arriere

Cette fonction affiche la liste des solutions possibles ainsi que leur description, et demande à l'utilisateur la solution qu'il souhaite tester jusqu'à ce qu'il en sélectionne une valide. La fonction fait ensuite un appel à la fonction faisant réellement un chainage arrière, puis affiche si la solution testée est la solution appropriée.

Passons alors à la fonction faisant réellement le chainage arrière, le but de cette fonction est de déterminer si le but donné en entrée (but) peut être déduit à partir des faits connus et des règles du système expert. Si certaines informations sont manquantes pour prendre une décision, elles seront demandées à l'utilisateur. Le premier appel à cette fonction sera typiquement de la forme :

```
(chainage_arriere (eq Solution solution_a_tester))
```

Afin de vérifier si la solution que veut tester l'utilisateur est appropriée vis-à-vis de son problème.

Pour commencer, la fonction vérifie d'abord si le but appartient déjà aux faits connus en utilisant la fonction appartient_aux_faits. Si c'est le cas, elle retourne T.

Sinon, la fonction commence une boucle qui parcourt les règles candidates (obtenues grâce à la fonction regles_candidates) une par une. Pour chaque règle, elle

vérifie si toutes les conditions de la règle sont vraies en utilisant la fonction chainage_arriere récursivement.

Si toutes les conditions sont vraies, elle ajoute la conclusion de la règle aux faits connus en utilisant la fonction ajouter_fait, puis elle marque la variable ok comme étant vraie pour indiquer que le but a été trouvé.

Si aucune règle n'a permis de trouver le but, la fonction vérifie la variable concernant le but défini dans la base des faits, et, si ce n'est pas cas, demande la valeur de la question à l'utilisateur en utilisant la fonction demander_valeur_question. La valeur est ensuite ajoutée aux faits connus. Ensuite, on vérifie à nouveau si le but peut être déduit à partir des faits connus. Enfin, la fonction retourne la valeur de la variable ok pour indiquer si le but a été trouvé ou non.

Voici le code des principales fonctions utilisées :

```
(defun appartient_aux_faits (fait)
  (let ((element (assoc (cadr fait) *faits*)))
    (if element
      (funcall (car fait) (cadr element) (caddr fait))
      NIL)))
```

FIGURE 10 – appartient_aux_faits

```
(defun regles_candidates (but regles)
  (if regles
    (if (equal (conclusion_regle (car regles)) but)
      (cons (car regles) (regles_candidates but (cdr regles)))
      (regles_candidates but (cdr regles)))))
```

FIGURE 11 – regles_candidates

```
(defun modifier_fait (variable valeur)
  (setf (cadr (assoc variable *faits*)) valeur))


(defun ajouter_fait (variable valeur)
  (if (assoc variable *faits*)
    (modifier_fait variable valeur)
    (pushnew (list variable valeur) *faits*)))
```

FIGURE 12 – ajouter_fait (et modifier_fait)

```
(defun demander_valeur_question (question)
  (let* ((element (assoc question *questions*))(valeur NIL)(
valeur_valide (caddr element)))
    (if element
      (progn
        (loop while (not (eval valeur_valide)) do
          (progn
            (format t (cadr element))
            (setq valeur (read))))

        (cond ((eq valeur 'y) T)
              ((eq valeur 'n) nil)
              (t valeur)))
      (progn
        (format t "Question inconnue : ~s ~&" (symbol-name question
))
        NIL))))
```

FIGURE 13 – demander_valeur_question

Cette fonction se sert d'une condition définie à l'avance pour permettre de s'assurer de la validité de la valeur saisie par l'utilisateur. En effet, lors de l'ajout d'une question, on spécifie une condition que devra remplir la réponse de l'utilisateur. Cette condition est vérifiée avec "(eval valeur_valide)", valeur_valide étant la condition spécifiée au moment de la création de la question. Voici le code de l'ajout d'une question :

```
(defun ajouter_question (question texte_question valeur_valide)
  (pushnew (list question texte_question valeur_valide) *questions
*))
```

FIGURE 14 – ajouter_question

Et voici des exemples d'ajouts de questions, avec la chaine de caractères et la condition associée. Nous avons ici trois exemples, un qui exige une réponse entre deux valeurs (possible ou impossible), un autre entre quatre valeurs (element_critique, peripherique, affichage ou energie), et un dernier qui exige que la réponse soit un entier positif ou nul :

```
(ajouter_question 'Demarrage "~&Le demarrage de votre ordinateur
est POSSIBLE ou IMPOSSIBLE ? " '(member valeur '(possible
impossible)))

(ajouter_question 'Domaine "~&Le probleme concerne quel partie de
votre ordinateur?~&ELEMENT_CRITIQUE ; PERIPHERIQUE ; AFFICHAGE ;
ENERGIE : " '(member valeur '(element_critique peripherique
affichage energie)))

(ajouter_question 'Nb_os "~&Combien d'OS possedez-vous ? " '(and (
integerp valeur) (>= valeur 0)))
```

FIGURE 15 – ajouter_question

Voici le code de la fonction correspondant au chainage arrière :

```
(defun chainage_arriere (but)
  (if (appartient_aux_faits but)
    T
    (let
      ((cand (regles_candidates but *regles*)) (ok NIL) (prem NIL)
(condition NIL) (regle NIL)(valeur NIL))

      (loop while (and cand (not ok)) do

        (setq regle (pop cand))
        (setq prem (conditions_regle regle))
        (setq ok T)

        (loop while (and prem ok) do
          (setq condition (pop prem))
          (setq ok (chainage_arriere condition)))

        (if ok
          (ajouter_fait (cadr (conclusion_regle regle)) (caddr (
conclusion_regle regle)))))

      (if (not ok)
        (if (not (assoc (cadr but) *faits*))
          (progn
            (setq valeur (demander_valeur_question (cadr but)))
            (if valeur
              (progn
                (ajouter_fait (cadr but) valeur)

                (setq ok (appartient_aux_faits but)))))))

      ok)))
```

FIGURE 16 – chainage_arriere

4 Test de nos moteurs d'inférences

Dans cette partie, nous allons vous présenter un exemple de fonctionnement de nos moteurs d'inférences.

4.1 Test du chainage avant

Voici les captures d'écran d'utilisation de notre système expert, nous avons choisi dans la fonction afficher-menu le chainage avant. Les deux captures vont présenter le début du chainage sans ajout de problème courant, et en cherchant la solution à un problème non courant :

```
CG-USER(102):
Voulez-vous faire un chainage avant ou arriere ? (avant/arriere) : avant
Voulez-vous completer les problemes courants ?
Votre choix (Y/N): n
**********************************************************
Commencons par voir les problemes les plus courants.
Votre probleme correspond a la description suivante :
Ton probleme est lie a ton Os, ton ordinateur demarre et est capable de faire des mises a jour
Cependant le probleme affecte les performances et/ou l'integrite de ton ordinateur malgre un redemarrage.(Y/N)
Votre choix : n
Votre probleme correspond a la description suivante :
Votre probleme est lie a un logiciel specifique qui ne fonctionne plus sur votre ordinateur.(Y/N)
Votre choix : n
Votre probleme correspond a la description suivante :
Ton probleme est materiel, empeche ton ordinateur de fonctionner pleinement.
Il est lie a l'energie et persiste apres un changement de chargeur.(Y/N)
Votre choix : n
Votre probleme correspond a la description suivante :
Votre ordinateur ne demarre pas et ne possede qu'un seul OS.(Y/N)
Votre choix : n
**********************************************************
Votre probleme ne semble pas etre un probleme courant.
Decrivez moi plus en details votre probleme.
*                                                       *
```

FIGURE 17 – Choix du chainage avant

```
Votre probleme est de type MATERIEL ou LOGICIEL ? materiel
Votre probleme materiel est de type PERFORMANCE ou FONCTIONNEMENT ? performance
Ce probleme diminue les performances de votre SYSTEME ou de votre connexion INTERNET ? systeme
**********************************************************
Voici la solution a votre probleme :
Reparez ou remplacez le composant faisant defaut.
Si vous n'avez pas les connaissances pour le faire, contactez un reparateur.
**********************************************************
```

FIGURE 18 – Résultat du chainage avant

Les deux captures vont présenter l'ajout de problème courant et révisions des probabilités de chacun de ces problèmes :

```
Voulez-vous completer les problemes courants ?
Votre choix (Y/N): y
Combien de problemes voulez vous ajouter a la liste des problemes courant ?
Votre choix : 1
**************** MAJ probleme courant ****************
Entrez le nom de la solution parmis cette liste :
REDEMARRER_MEMOIRE
ANALYSE_ANTIVIRUS
GESTION_ERREUR_INTERNE
ACHETER_ORDI
INSTALLATION_BOOTLOADER_MULTIPLE
INSTALLATION_BOOTLOADER
INSTALLATION_OS
MAJ_REINSTALLATION
LOGICIEL_ALTERNATIF
SIGNALER
INSTALLATION_PILOTE
REMPLACEMENT_PORT
REMPLACEMENT_PERIPH
REMPLACEMENT_ECRAN
MAJ_CARTE_GRAPHIQUE
REMPLACEMENT_CHARGEUR
REMPLACEMENT_BATTERIE
REMPLACEMENT_REPARATEUR
CONTACT_OPERATEUR
Votre saisie :installation_pilote
Entrez la description du probleme : votre ordinateur ne semble plus etre en mesure de communiquer
La somme des probabilites doit être egale a 1.
Veuillez verifier les probabilites des problemes deja enregistres et entrer de nouvelles probabilites si necessaire.
Somme proba actuelle : 0.000
Reste : 1.000
Entrez la probabilite du probleme dont la solution est 'INSTALLATION_PILOTE'
Votre saisie (entre 0 et 1): 0.03
Entree non valide, veuillez entrer un nombre entre 0 et 1
Somme proba actuelle : 0.030
Reste : 0.970
Entrez la probabilite du probleme dont la solution est 'ANALYSE_ANTIVIRUS'
Votre saisie (entre 0 et 1): 0.4
```

FIGURE 19 – Ajout d'un problème aux problèmes courants

```
La base des problèmes les plus courant a ete mise a jour
**************************************************
**********************************************************************
Commencons par voir les problemes les plus courants.
Votre probleme correspond a la description suivante :
```

FIGURE 20 – Ajout d'un problème aux problèmes courants

4.2 Test du chainage arrière

Voici les captures d'écran d'utilisation de notre système expert, dans le cas où nous choisissons le chainage arrière dans la fonction afficher-menu. Les captures vont présenter le choix du problème à tester, suivi du comportement du système expert dans le cas ou la solution testée est la bonne et dans le cas contraire.

```
CG-USER(103): (afficher-menu)
Voulez-vous faire un chainage avant ou arriere ? (avant/arriere) : arriere

Quelle solution vous semble appropriee pour le probleme que vous avez ?

REDEMARRER_MEMOIRE :
Redemarrez votre ordinateur.
Un redemarrage de temps en temps permet de nettoyer la memoire et de fermer les processus qui tournent dans le vide

ANALYSE_ANTIVIRUS :
Le probleme peut etre lie a des fichiers corrompus ou a un programme malveillant.
Dans tous les cas, lancez une analyse avec les differents logiciels dont vous disposez et ceux mis en place par votre systeme d'exploitation.
```

FIGURE 21 – Choix du chainage arrière

```
Votre choix : signaler
Votre probleme est de type MATERIEL ou LOGICIEL ? logiciel
Quel logiciel est touche, votre OS ou un LOGICIEL specifique ? logiciel
Votre probleme est GLOBAL ou, vous est propre, UNIQUE ? global
La solution SIGNALER est la bonne
NIL
```

FIGURE 22 – La solution choisie est adaptée

```
Votre choix : signaler
Votre probleme est de type MATERIEL ou LOGICIEL ? logiciel
Quel logiciel est touche, votre OS ou un LOGICIEL specifique ? os
La solution SIGNALER n'est pas la bonne
NIL
```

FIGURE 23 – La solution choisie n'est pas adaptée

5 Répartition du travail

5.1 Travail réalisé ensemble

Il est important de travailler de manière équitable en équipe, car cela peut renforcer la coopération, aider à éviter les tensions et les conflits et être bénéfique pour le projet lui-même. C'est pourquoi nous devons être conscients de cet aspect et faire en sorte de travailler de cette manière pour atteindre nos objectifs communs.

Voici une liste des tâches que nous avons réalisées en commun :

1. *ajouter_fait* : permet d'ajouter ou de mettre à jour un fait dans la base de connaissances.
2. *modifier_fait* : fonction appelée par ajouter_fait en cas d'élément déjà présent dans la base de faits
3. *regles_candidates* : prend en entrée une cible (but) et une liste de règles (regles) et retourne une liste de règles dont la conclusion est égale à la cible.
4. Construction de l'arbre du système expert
5. Rédaction des questions posées à l'utilisateur, permettant ensuite la récupération de saisie et d'itérer sur nos deux chainages
6. Débogage sur tout le code

Chacun de nous a contribué de manière significative à ces tâches et nous avons travaillé ensemble de manière efficace pour les accomplir.

5.2 Travail réalisé par etudiant1

Dans le cadre de ce TP, j'ai travaillé sur le chainage arrière de notre système expert et la base de règles nécessaire pour nos différents chainages. J'ai également créé les fonctions d'usages nécessaires à nos moteurs d'inférences.

Voici une liste tâches que j'ai réalisées :

1. *appartient_aux_faits* : vérifie si un fait donné est présent dans la base de connaissances en comparant les termes du fait à l'aide d'un opérateur de comparaison spécifié.
2. *demander_valeur_question* : demande à l'utilisateur de fournir une valeur pour une question donnée et retourne la valeur saisie une fois qu'elle a été validée. En plus d'être associée à une chaine de caractères, chaque question est associée à une condition qui permet de vérifier si la valeur saisie par l'utilisateur est correcte.
3. *ajouter_regle* : permet d'ajouter une règle à la base de règle
4. *chainage-arriere* : Fonction servant à faire saisir par l'utilisateur la solution à vérifier. Une fois la solution vérifiée, la fonction réalisera un affichage pour indiquer les résultats.
5. *chainage_arriere* : Fonction récursive appelée par chainage-arriere pour procéder à la vérification d'une solution.
6. *afficher_menu* : Menu principal permettant de choisir quelles parties du programme, on souhaite utiliser.

J'ai œuvré à rendre simples les conditions de validité associées aux questions de sorte à ne pas avoir à s'en soucier lors de leur appel. Ces conditions, par exemple "(and (integerp valeur) (>= valeur 0))", ont nécessité la création d'une variable globale "valeur", sinon cette dernière n'était pas reconnue lors de l'évaluation de la condition, même si "valeur" existait dans un contexte local.

5.3 Travail réalisé par etudiant2

Dans le cadre de ce TP sur les systèmes expert, j'ai été chargé de mettre en place le chainage avant et les questions posé à l'utilisateur. Pour ce faire, j'ai développé plusieurs fonctions qui ont été utilisées dans le processus d'inférence permettant de trouver la solution la plus adaptée.

Voici une liste des tâches que j'ai réalisées :

1. *get-description-val-regles* : me permettant d'obtenir la liste des règles qui suive la règle actuelle, cela se fait grâce, à la valeur prise par la variable Description.
2. *get-nom-var-regles* : qui permet d'obtenir le nom de la variable permettant de vérifier la règle suivante.
3. *find-next-regles* : permettant d'obtenir la liste des règles suivant un état donné.
4. *chainage-avant* : qui va effectuer le chainage avant en posant dans un premier temps les questions concernant les problèmes les plus courants (les plus probables en premiers). Puis, dans un second temps, si le problème n'a pas été trouvé, va questionner l'utilisateur afin de lui proposer la solution adaptée à son problème.
5. Ajout dans le menu de la possibilité de compléter la liste des problèmes les plus courants
6. *demander-nom* et *ajouter-pb-courant* qui permettent d'ajouter un problème, sa probabilité et sa description à la liste des problèmes courants et demandant à l'utilisateur de saisir toutes les probabilités afin que la somme de 1.

J'ai travaillé sur le bonus permettant, dans le chainage avant, de répondre facilement à l'utilisateur si son problème fait partie des problèmes courants. La liste contient les noms des solutions, ainsi que la probabilité et la question associée. Cette fonctionnalité permet de rassurer l'utilisateur ayant un problème commun et ne connaissant pas l'ensemble des caractéristiques de son problème.

Conclusion

Le troisième TP nous a permis de mettre en pratique nos connaissances sur les systèmes experts et le chainage avant et arrière. Nous avons développé un programme appelé Jarvis Solution, qui est capable de trouver des solutions efficaces à des problèmes informatiques grâce à ces méthodes de représentation des connaissances.

Ce TP nous a donné l'occasion de travailler sur un sujet qui nous passionnait et de développer notre créativité et notre capacité à résoudre des problèmes de manière autonome. Nous avons apprécié cette expérience de travail et nous sommes satisfaits du résultat obtenu avec Jarvis Solution.

Plusieurs pistes d'amélioration pourraient être mises en place pour notre système expert. Tout d'abord, nous pourrions développer une interface graphique plus dynamique pour l'utilisateur, afin de rendre l'expérience plus agréable et intuitive. Ensuite, nous pourrions également proposer un plus grand nombre de règles, afin de rendre le système plus complet et de répondre aux besoins de tous les utilisateurs. Enfin, nous pourrions également mettre en place des moteurs d'inférence de recherche en largeur. Ainsi, un utilisateur aurait face à lui de nombreuses solutions à tester pour résoudre son problème.

Table des figures

SOURCES

1. https ://www.cybermalveillance.gouv.fr/tous-nos-contenus/actualites/comment-identifier-et-supprimer-un-virus : pour la partie malware, virus
2. : https ://www.cybermalveillance.gouv.fr/tous-nos-contenus/actualites/pourquoi-est-il-dangereux-de-negliger-les-mises-a-jour : pour bien comprendre l'importance de maintenir à jour ses logiciels et son OS
3. : https ://www.facilitoo.fr/2017/10/07/problemes-informatiques-courants/ : pour les problèmes courants