IA01 : Rapport du TP3

Auteurs :
Camille Gerin-Roze
Thomas Perrin

Le 3 Janvier 2016

Table des matières

Introduction

Chapitre 1

Connaissances nécessaires au système expert

1.1 Paramètres à fournir au système

Afin de déterminer le nombre de nains nécessaire à un chantier, plusieurs paramètres seront nécessaire :

— La longueur du tunnel
— Sa hauteur
— Sa largeur
— Le nombre de jours maximum
— Le type de roche
— Le type de pioche employé

Avec ces paramètres, on va pouvoir déterminer si oui ou non le chantier est réalisable. Si il l'est, on obtiendra un nombre total de nains avec les différents rôles de chacun. Un indicateur sera présent pour dire si une équipe de nuit est nécessaire ou non pour compléter le chantier en temps et en heures.

1.2 Traduction des connaissances en règles

La base de règles est décomposé en plusieurs parties :

— Les règles du type de roche (RTR) Ces règles vont permettre, en fonction du type de roche, de fixer la vitesse de travail d'un nain par jour.
— Les règles du type de pioche (RTP) Ces règles vont ajouter un coefficient multiplicateur à la vitesse de base, pour aller plus ou moins vite en fonction des moyens de l'employeur.
— Les règles concernant la réalisation du chantier (RCR) Ces règles vont déterminer si le chantier est réalisable avec les paramètres énoncés. En effet, en fonction de la vitesse des nains, de la hauteur, largeur et longueur du tunnel, on peut d'avance vérifier si cela tiendra avec le nombre de jours énoncé ou non.
— Les règles calculant le nombre de nains (RN) Ces dernières règles vont conclure en calculant le nombre de nains totals.

Toutes les règles ont été déterminés par la vidéo présentés ci-dessus, et par des échanges avec John Lang, le créateur de l'univers du Donjon de Naheulbeuk. Nous avons seulement modifié une seule règle pour la rendre cohérente, celle des nains de ravitaillements en bière.

Chapitre 2

Implémentation du système expert

2.1 Choix de la représentation Lisp

2.1.1 Base de règles

Pour représenter les règles de notre SE, nous avons choisi d'utiliser une liste qui contiendra 2 sous-listes (les nouveaux faits et les prémisses) ainsi que le nom de la règle. Par exemple, voici la règle RCR1 qui permet de savoir si un chantier est réalisable :

```
(
        (  ;;;;  DEBUT DES NOUVEAUX FAITS
                (ChantierRealisable . T)
                (EquipeDeNuit . NIL)
        );;;;  FIN DES NOUVEAUX FAITS
                (;;;;DEBUT DES PREMISSES
                (COMPARAISON (>=
                        (* LargeurTunnel HauteurTunnel
                            NombreDeJours VitesseNain)
                        LongueurTunnel
                ))
        );;;;FIN DES PREMISSES
RCR1);;;;NOM DE LA REGLE
```

On peut noter que les premisses ont un format bien spécial. En effet, chaque condition commence par le type de vérification à faire. Cela peut être une égalité, une comparaison, ou encore une définition (savoir si un fait a déjà été défini). Cette étiquette permet de simplifier le traitement lors de la vérification des prémisses. Pour les prémisses, une fonction va donc vérifier si LargeurTunnel * HauteurTunnel * NombreDeJours * VitesseNain est bien supérieur ou égale à la longueur du tunnel. Si c'est le cas, notre moteur d'inférence va apeller une nouvelle fonction qui va évaluer les valeurs des nouveaux faits et les ajouter à notre base de fait. Nous avons également une A-List qui contient la description

de chaque règles, mais celle-ci a été mise à part dans le but de garder un code clair. Cette représentation permet non seulement d'avoir un format homogène pour toutes les règles, et donc de traiter chaque règle de la même façon, mais elle permet également de traiter des conditions complexes qui demandent l'évaluation de plusieurs opérations ou encore des opérations qui demandent de rechercher dans la base de faits.

2.1.2 Base de faits

La base de faits a été faites de la manière la plus simple possible, celle-ci est juste une liste donc chaque membre est une sous-liste qui représente un fait. Un fait est tout simplement constitué d'une étiquette, donc le nom du fait, et d'une valeur qui peut être un nombre, un symbole, ou même une valeur booléenne.

```
(
  (VitesseNain 3)
  (TypeRoche GRANITE)
  (LargeurTunnel 3)
  (LongueurTunnel 28)
  (HauteurTunnel 1)
)
```

Ici on peut voir qu'il y a 4 faits dans notre base de faits : la vitesse des nains, le type de roche, la largeur et la longueur du tunnel, et la hauteur du tunnel.

2.2 Le moteur d'inférences

Le moteur d'inférence est un moteur à chaînage avant avec un fonctionnement simple, mais pour bien comprendre comment il fonctionne, il faut expliciter certaines fonctions qui permettent de savoir si les prémisses sont satisfaits ou bien d'ajouter les faits à notre base de faits. Nous allons donc commencer par présenter toutes ces fonctions.

2.2.1 Validation des prémisses

Pour que les prémisses soient validés, il faut que toutes les conditions soient validées. Donc nous allons itérer sur toutes les conditions de la liste des prémisses pour vérifier qu'elles sont toutes validée. Mais certaines conditions sont complexes et demandent d'aller vérifier l'existence ou la valeur de faits dans la base de faits. Pour cela nous avons défini une fonction evalOperande qui va nous permettre d'évaluer la valeur de chaque opérande :

```
(defun evalOperande(operande)
  (let ((newList (list)))
      (dolist (x operande newList)
        (cond
          ((LISTP x)
            (setq newList (append newList (list (evalOperande x)))))
          ((OR (EQUAL x '*) (EQUAL x '/) (EQUAL x '-) (EQUAL x '+))
            (setq newList (APPEND newList (list x))))
          ((SYMBOLP x)
            (if (NULL (ASSOC x *BaseFaits*))
              ;;;; SI LE SYMBOLE N'EST PAS DEFINI DANS LA BDF
                (return-from evalOperande NIL)
              (setq newList (APPEND newList (cdr (ASSOC x *BaseFaits*))))
            ))
          (T (setq newList (APPEND newList (list x))))
        )
    )
  )
)
```

Pour évaluer un opérande, on va vérifier si celui-ci est une liste, si c'est une liste on va alors appeller notre fonction récursivement sur chaque éléments de la liste. Dans le Cas où c'est un symbole, on va alors aller chercher sa valeur dans la base de faits. Si jamais la valeur n'est pas dans la base de faits, il est important que la fonction renvoie NIL pour qu'on sache que cela ne peut pas être évalué et que la condition n'est pas respectée. À la fin de la fonction, on aura une liste qui pourra être évaluée grâce à un simple eval.
Ensuite il y a deux autres fonctions, une qui va vérifier si une condition est respectée, et une qui vérifiera si toutes les conditions sont vérifiées. La fonction qui vérifie si une condition est respectée va d'abbord vérifier l'étiquette placée au début de celle-ci (COMPARAISON, DEFINI, EGALITE) pour savoir comment elle doit la traiter.

```
(defun conditionRespecte?(condition)
(cond
  ((EQUAL (car condition) 'EGALITE)
      (return-from conditionRespecte?
          (EQUAL (cadr (ASSOC (car (cadr condition)) *BaseFaits*))
           (cadr (cadr condition)))))
  ((EQUAL (car condition) 'DEFINI)
      (return-from conditionRespecte?
          (NOT (NULL (ASSOC (car (cadr condition)) *BaseFaits*)))))
  ((EQUAL (car condition) 'COMPARAISON)
    (return-from conditionRespecte?
      (eval
        (let ((expression))
          (dolist (x (cadr condition) expression)
            (cond
              ((estUnComparateur? x)
                (setq expression (append expression (list x))))
                ((listp x)
                  (if (evalOperande x)
                    (setq expression
                      (append expression (list (eval (evalOperande x))))
                    )
                      NIL
                    ))
                (T
                  (if (ASSOC x *BaseFaits*)
                    (setq expression
                      (append expression (list (cadr (ASSOC x *BaseFaits*))))
                     )
                    NIL
                    ))
            )
          )
        ))))
  )
)


(defun premisseRespecte?(regle)
  (let ((test T))
    (dolist ( x (getPremisse regle) test)
      (setq test (AND test (conditionRespecte? x)))
    )
  )
)
```

Un fois que la condition est vérifiée, on peut tout simplement faire une boucle sur toutes les conditions pour savoir si celles-ci sont toutes vrais, si elles sont toutes vrais on pourra ajouter les nouveaux faits à notre base de fait.

2.2.2 Calcul des nouvelles valeurs

Dans notre système expert, il est possible qu'une variable soit mises à jour dans certains conditions. Il est aussi possible qu'il faille calculer la valeur de la variable de notre système expert, donc l'ajout d'un nouveau fait n'est pas si aisé.

```
(defun ajouterFait(fait)
  (if (assoc (car fait) *BaseFaits*)
    (setq *BaseFaits* (remove (car fait) *BaseFaits* :key #'first))
    (push fait *BaseFaits*)
)

(defun evaluerValeur(val)
  (cond
    ((OR (EQUAL val T) (NULL val)(NUMBERP val)) val)
    ((SYMBOLP val) (cadr (ASSOC val *BaseFaits*)))
    ((LISTP val)
      (let (newVal)
          (dolist (x val newVal)
            (if (OR
                  (EQUAL x '+) (EQUAL x '-) (EQUAL x '/)
                  (EQUAL x '*) (EQUAL x 'truncate) (EQUAL x 'ceiling)
                )
              (setq newVal (APPEND newVal (list x)))
              (setq newVal (APPEND newVal (list (evaluerValeur x))))
            )
          )
        )
    )
  )
)

(defun majBDF(regle)
  (dolist (x (getNouveauxFaits regle) NIL)
    (if (listp (evaluerValeur (cdr x)))
      (ajouterFait (list (car x) (eval (evaluerValeur (cdr x)))))
      (ajouterFait (list (car x) (cdr x)))
    )
  )
)
```

La fonction evaluerValeur va couvrir plusieurs cas :

- Dans le cas où la valeur passée en paramètre est un nombre ou une valeur logique (T ou NIL) alors on ne fait rien
- Dans le cas où la valeur est un symbole, on va aller chercher sa valeur dans la base de faits
- Et dans le cas où la valeur est une liste, alors on va créer une nouvelle expression évaluable en allant chercher toutes les valeurs dont on a besoin dans la base de faits.

En suite, la mise à jour de la base de faits se fait en évaluant notre nouvelle expression dans le cas où elle a besoin d'être évaluée.

2.2.3 Moteur à chaînage avant

```
(defun chainageAvant()
  (let ((listeRegle *BaseRegles*)(nouveauxFaits T)(reglesUtilise NIL))
  (loop while nouveauxFaits do
    (setq nouveauxFaits NIL)
    (dolist (r listeRegle NIL) ;;;; TANT QU'IL Y A DES R GLES     TESTER
          (format t "~%REGLE=~a~%BDF=~a" (caddr r) *BaseFaits*)
            (if (premisseRespecte? r) ;;;; SI LES PREMISSES DE LA REGLE SONT RESPECTES
                (progn
                    (setq nouveauxFaits T) ;;;; ON NOTIFIE QU'IL Y A EUT UN NOUVEAU FAIT

                    ;;;; ON REITIRE LA REGLE DE LA LISTE
                    (setq listeRegle (remove (caddr r) listeRegle :key #'third))
                    (majBDF r)
                    (push (caddr r) reglesUtilise)
                )
            )
      )
  )
  reglesUtilise
)
)
```

Le moteur d'inférence est un moteur d'inférence en chaînage avant et en profondeur d'abbord car on ajoute les faits directement après qu'ils aient été déterminés par une règle. On notera également que le moteur de chainage avant va renvoyer la liste des règles utilisées, mais la base de fait étant une variable globale celle-ci est directement mise à jour.

2.2.4 Test du moteur d'inférences

Pour pouvoir utiliser notre système expert nous avons créé une interface utilisateur, voici un exemple d'utilisation du système expert avec les données du problème exposé par John Lang. On peut remarquer que le résultat est légèrement différent, ceci est dû au fait qu'en correspondant avec lui, nous avons eu de nouvelles informations qui nous ont permis de d'enrichir le système expert, d'où la petite différence de résultat.

```
De quel longueur est votre tunnel ?28
De quel largeur est votre tunnel ?3
Quel est la hauteur de votre tunnel ?1
Dans combien de temps votre tunnel doit   tre   pr t ?2
Selectionner le type de roche dans lequel vous voulez creuser : GRANITE

... DEROULEMENT DE TOUTES LES REGLES ...

_____________________________________________________________________
_____________      LE CHANTIER EST REALISABLE      _____________________
_____________________________________________________________________

VOIL  LA COMPOSITION DE VOS EQUIPES POUR FAIRE VOTRE CHANTIER LE
PLUS RAPIDEMENT POSSIBLE :

VOUS AVEZ BESOIN D'UNE EQUIPE DE JOUR COMPOSE DE :
Nains mineurs : 2
Nains Guerisseurs : 1
Nains Forgerons : 1
Nains Tourneurs de manche : 1
Nains Ravitailleurs : 8
Nains Plongeurs (Vaisselle) : 2

AINSI QUE D'UNE EQUPIDE DE NUIT COMPOSE DE :
Nains Mineurs : 2
Nains Guerisseurs : 1
Nains Forgerons : 1
Nains Tourneurs de manche : 1
Nains Ravitailleurs : 8
Nains Plongeurs (Vaisselle) : 2
Nains Surveillants : 4
Nains Managers : 2
Nains Porteurs de lanternes : 4

TOTAL :                                                    40 nains
_____________________________________________________________________


(RN3 RN1 RCR2 RTP4 RTR2)
```

Chapitre 3

Scénarios d'utilisation

3.1 Chantier non réalisable

Les nains n'apprécient pas qu'on leur marche sur les pieds, donc il faut faire attention de ne pas mettre trop de nain dans un tunnel qui n'est pas assez large. On va faire tourner notre moteur d'inférence avec les données suivantes :

- Largeur du tunnel : 2
- Longueur du tunnel : 100
- Hauteur du tunnel : 1
- Temps maximale : 2
- Type de roche :

Avec ces données, on a le résultat suivant :

BDF =
((EQUIPEDENUIT NIL) (CHANTIERRALISABLE NIL) (VITESSENAIN 3)
(TYPEDEPIOCHE STANDARD) (TYPEDEROCHE GABBROS) (HAUTEURTUNNEL 1)
(NOMBREDEJOURS 2) (LONGUEURTUNNEL 100) (LARGEURTUNNEL 2))

———— LE CHANTIER EST N'EST PAS REALISABLE ————

(RCR3 RTP4 RTR3)

On remarque que la vitesse des nains est de 3 à cause de la roche, Mais le chantier n'est pas réalisable. Les trois règles utilisées sont :

- RCR3 : Si

  $$LongueurTunnel < LargeurTunnel * hauteurTunnel * nbJourMax * VitesseNain * 2$$

  alors le chantier n'est pas réalisable
- RTP4 : Si la pioche est standard, alors la vitesse du nain est multipliée par 1
- RTR3 : Si la roche est de type Gabbros, alors le nain avancera avec une vitesse de 3.

Donc il est tout à fait normal que le chantier soit irréalisable, car il faudrait que le tunnel soit beaucoup plus large pour pouvoir y mettre plus de nains.

3.2 Chantier réalisable sans équipe de nuit

Pour le chantier réalisable sans équipe de nuit il faut un tunnel assez large pour pouvoir accueuillir beaucoup de nain mineurs, donc nous allons tester avec les valeurs suivantes :

- Largeur du tunnel : 6
- Longueur du tunnel : 28
- Hauteur du tunnel : 1
- Temps maximale : 2
- Type de roche : Granite

```
  BDF =
((NAINSCALCUL T) (NBNAINTOTAL 30) (NBNAINPLONGUEUR 4)
 (NBNAINRAVITAILLEMENT 16) (NBNAINTOURNEURMANCHE 2) (NBNAINFORGERON 2)
 (NBNAINGUERISSEUR 2) (NBNAINMINIER 4) (EQUIPEDENUIT NIL)
 (CHANTIERRALISABLE T) (VITESSENAIN 3) (TYPEDEROCHE GRANITE) (HAUTEURTUNNEL 1)
 (NOMBREDEJOURS 2) (LONGUEURTUNNEL 28) (LARGEURTUNNEL 6) (NBNAINMANAGER 2)
 (NBNAINSURVEILLANT 4) (NBNAINPORTEURLANTERNE 4) (TYPEDEPIOCHE STANDARD))
```

LE CHANTIER EST REALISABLE

VOIL LA COMPOSITION DE VOS EQUIPES POUR FAIRE VOTRE CHANTIER LE PLUS RAPIDEMENT POSSIBLE :

VOUS AVEZ BESOIN D'UNE EQUIPE DE JOUR COMPOSE DE :
Nains mineurs : 4
Nains Guerisseurs : 2
Nains Forgerons : 2
Nains Tourneurs de manche : 2
Nains Ravitailleurs : 16
Nains Plongeurs (Vaisselle) : 4

TOTAL : 30 nains

(RN2 RN1 RCR1 RTP4 RTR2)

- RN2 : Si il n'y a pas d'équipe de nuit et que l'équipe de jour est commencé, on complète l'équipe de jour avec les nains de ravitaillement et les nains plongueurs.
- RN1 : Si le chantier est réalisable, alors on crée le début de l'équipe de jour qui sera présente dans tous les cas (nain miniers, nains guérisseurs, nains tourneurs de manche et nains forgeron).
- RCR1 : Si LongueurTunnel inferieur à LargeurTunnel*hauteurTunnel*nbJourMax*VitesseNain, le chantier est réalisable et nous n'avons pas besoin d'équipe de nuit
- RTP4 : Si la pioche est standard, alors la vitesse du nain est multipliée par 1
- RTR2 : Si la roche est de type Granite, alors le nain avancera avec une vitesse de 3.

3.3 Chantier réalisable avec une équipe de nuit

Pour le chantier réalisable avec une équipe de nuit nous allons utiliser les mêmes valeurs que le problème originel et que nous avons utilisé dans le chapitre 2.

- Largeur du tunnel : 3
- Longueur du tunnel : 28
- Hauteur du tunnel : 1
- Temps maximale : 2
- Type de roche : Granite

```
--------------------------------------------------------------------------------
----------------------  LE CHANTIER EST REALISABLE  ----------------------
--------------------------------------------------------------------------------

VOIL  LA COMPOSITION DE VOS EQUIPES POUR FAIRE VOTRE CHANTIER LE
PLUS RAPIDEMENT POSSIBLE :

VOUS AVEZ BESOIN D'UNE EQUIPE DE JOUR COMPOSE DE :
Nains mineurs : 2
Nains Guerisseurs : 1
Nains Forgerons : 1
Nains Tourneurs de manche : 1
Nains Ravitailleurs : 8
Nains Plongeurs (Vaisselle) : 2

AINSI QUE D'UNE EQUPIDE DE NUIT COMPOSE DE :
Nains Mineurs : 2
Nains Guerisseurs : 1
Nains Forgerons : 1
Nains Tourneurs de manche : 1
Nains Ravitailleurs : 8
Nains Plongeurs (Vaisselle) : 2
Nains Surveillants : 4
Nains Managers : 2
Nains Porteurs de lanternes : 4

TOTAL :                                                    40 nains
--------------------------------------------------------------------------------
```

(RN3 RN1 RCR2 RTP4 RTR2)

- RN3 : Si une équipe de nuit est nécessaire et que l'équipe de jour est commencé, on crée l'équipe de nuit (Nains porteur de lanterne, nains surveillant, nains manager), on double l'effectif de l'équipe de jour pour avoir l'équivalence en équipe de nuit et on termine par les nains de ravitaillement et les nains plongueurs.
- RN1 : Si le chantier est réalisable, alors on crée le début de l'équipe de jour qui sera présente dans tous les cas (nain miniers, nains guérisseurs, nains tourneurs de manche et nains forgeron).
- RCR2 : Si LargeurTunnel*hauteurTunnel*nbJourMax*VitesseNain ¡= LongueurTunnel ¡= LargeurTunnel*hauteurTunnel*nbJourMax*VitesseNain*2 , le chantier est réalisable et nous avons besoin d'une équipe de nuit
- RTP4 : Si la pioche est standard, alors la vitesse du nain est multipliée par 1
- RTR2 : Si la roche est de type Granite, alors le nain avancera avec une vitesse de 3.

Les résultats sont très proches de ceux exposés par John Lang dans la vidéo qui nous a permis de faire la plus grande partie des règles, mais il est normal qu'il y ait une légère différence car nous prenons en comptes d'autres données en compte. De plus nous avons correspondu avec John Lang, ce qui nous a forcé à modifier des règles. Les règles qui ont été utilisée sont cohérentes et correspondent bien aux données originels du problème (le type de pioche se modifie dans les configurations de l'UI et est mis à standard par défaut)

Conclusion