Compte rendu de TP INFO 805

Réalisé par :

Arnaud GUIGUE BILLON

Méwen COSSON

Jonathan DUMONT

Domaine d’application de notre système

Le domaine d’application de notre système est sur le jeu “League of legends” qui est un jeu sur lequel deux équipes de 5 joueurs s’affrontent. Chaque joueur contrôle un personnage qu’il choisit en début de partie. Les personnages du jeu possèdent plusieurs particularité :

- Le poste : représente l’endroit où le joueur va jouer sur la carte
- Le rôle : c’est la façon dont le personnage est jouer par exemple il y a le rôle “Tank”, il va prendre les coups pour son équipe, le rôle “support” qui lui va s’occuper de la soigner ou encore le rôle “combattant” qui lui va s’occuper de faire des dégâts à l’équipe ennemie.
- La difficulté : un chiffre qui représente la facilité ou la difficulté de la prise en main du personnage
- La portée : les personnages peuvent toucher leurs ennemies de deux façon soit au corps à corps (”Melee”) soit à distance (”Range”)
- Le prix : le coût d’achat du personnage
- La toxicité : si le joueur adverse va trouvé difficile de jouer contre ce personnage

Dans “League of legends”, il y a aujourd’hui plus de 150 champions et il peux être difficile pour un nouveau joueur quel champion lui correspondrait le mieux en fonction de ses goût et de son style de jeu. Le but de notre système est donc de recommandé un personnage à joueur à l’utilisateur. Pour cela le système va lui demandé différente caractéristique sur le champions qui sont : le poste, le rôle, la difficulté, la portée le prix et la

toxicité. Une fois que le système est informé de ces caractéristique celui-ci va pouvoir renvoyer un champion à jouer à l’utilisateur.

Déroulement

L’utilisateur va être invité à répondre à des questions. Après chaque réponse, il aura un retour des champions qu’il pourra potentiellement jouer

```
La lane ? (Top | Jungle | Mid | Bot | Support) : Top

"Galio"
"Dr. Mundo"
"Cho'Gath"
"Yorick"
"Renekton"
"Maokai"
"Teemo"
"Shen"
"Riven"
"Poppy"
"Nasus"
"Mordekaiser"
"Malphite"
"Jax"
"Darius"
```

```
Le role ? (Combattant | Tank | Assassin | Mage | Suppport | Shooter) : Tank

"Darius"
"Malphite"
"Mordekaiser"
"Nasus"
"Poppy"
"Shen"
"Maokai"
"Yorick"
"Cho'Gath"
"Dr. Mundo"
"Galio"
```

```
La range ? (Range | Melee) : Melee

"Galio"
"Dr. Mundo"
"Cho'Gath"
"Yorick"
"Maokai"
"Shen"
"Poppy"
"Nasus"
"Mordekaiser"
```

```
"Malphite"
"Darius"
```

```
La difficulté ? (1 | 2) : 2

"Darius"
"Malphite"
"Mordekaiser"
"Nasus"
"Poppy"
"Shen"
```

```
Le cancer : 3

"Malphite"
```

Et enfin le résultat!

```
vous allez jouer Malphite
```

Explication du code

Définition d’un champion

Nous avons créé un type champion, avec un nom, un post (Top, Jungle, Mid, Bot, Support), un role (Combattant, Tank, Assassin, Mage, Suppport, Shooter), une range (Range, Melee), et prix (3150, 4800).

```
(defclass $champion()(
    ($nom :accessor nom :initarg :nom :type string)
    ($role :accessor role :initarg :role)
    ($post :accessor post :initarg :post :type string)
    ($difficulte :accessor difficulte :initarg :difficulte :type string)
    ($range :accessor range :initarg :range :type string)
    ($prix :accessor prix :initarg :prix :type string)
    ($cancer :accessor cancer :initarg :cancer :type string)
))
```

Puis après cela, nous instancions l’objet et le mettons dans une variable.

```
(defvar darius (make-instance '$champion :nom "Darius" :role '("Support" "Tank") :post "Top" :difficulte "2"
    :range "Melee" :prix "4800" :cancer "5"))
(defvar draven (make-instance '$champion :nom "Draven" :role '("Shooter") :post "Bot" :difficulte "2"
    :range "Range" :prix "4800" :cancer "8"))
(defvar jax (make-instance '$champion :nom "Jax" :role '("Combattant") :post "Top" :difficulte "2"
    :range "Melee" :prix "4800" :cancer "3"))
```

Nous ajoutons ensuite tout les objets dans une liste.

```
(defvar liste-de-champion (list darius draven jax))
```

Base de fait

Pour notre base de fait, nous avons utilisé une structure composée d’un nom, d’une condition ainsi que d’une action.

```
(defstruct rule
  name
  conditions
  actions
)
```

Nous avons cinq faits

- nice-post : vérifie que tous les champions de la liste possèdent le poste souhaité par l’utilisateur
- nice-difficulte : vérifie qu’ils possèdent la bonne difficulté
- nice-range : vérifie qu’ils possèdent la bonne range
- nice-role : vérifie qu’ils possèdent le bon rôle
- nice-toxi : vérifie qu’ils possèdent le bon niveau de toxicités. Nous l’utilisons car plusieurs personnages peuvent avoir les mêmes autres informations, cela permet de ne retourner qu’un seul candidat.

Un fait s’organise de la sorte :

Condition : le joueur n’a pas donné l’information (dans le cas de l’exemple, le poste désiré)

Action : le programme demande au joueur le poste souhaité, puis retire de la liste tout les autres champions qui ne suivent pas ces critères. Après cela, nous affichons le nom de ceux qui restent dans la liste, afin d’avoir un suivi.

```
(defvar nice-post
    (make-rule
        :name 'nice-post
        :conditions (lambda () (equal poste nil))
        :actions (lambda ()
            (princ "La lane ? (Top | Jungle | Mid | Bot | Support) : ")
            (finish-output)
            (setq poste (read-line))
            (tri-post liste-de-champion '() )
            (displayN  liste)
            (terpri)
        )
    )
)
```

Le fait final, si le joueur a donné toutes les informations, cela veut qu’il reste, dans la liste, un ou aucun champion, s’il y en a un, nous affichons son nom, sinon nous informons l’utilisateur qu’aucun résultat n’a été trouvé.

```
(defvar finaly
    (make-rule
        :name 'nice-post
        :conditions (lambda () (and (not (equal range nil))
                                    (and (not (equal role nil))
                                        (and (not (equal difficulte nil))
                                            (and (not (equal cancer nil))
                                                 (not (equal poste nil))
                                                    )))))
        :actions (lambda ()
           (if (not (equal (car liste) nil))
                 (format t "vous allez jouer ~a.~%" (nom (car liste)))
                 (format t "aucun survivant"))
        )
    )
)
```

Puis nous ajoutons tout nos faits dans une liste de fait pour que notre moteur d’inférences puisse les travailler.

```
(defparameter lr (list finaly nice-toxi nice-difficulte nice-range nice-role nice-post ))
```

Fonctions utilitaires

Cette fonction nous permet de savoir si un élément est dans une liste.

```
(defun estDansListe (liste element)
    (if (null liste)
        nil
        (if (equal (car liste) element)
            t
            (estDansListe (cdr liste) element)
        )
    )
)
```

Cette fonction retourne la liste de champions qui valident la règle. Il en existe quatre autre similaire, pour les autres faits.

```
(defun tri-post (lc lt)
     (cond
        ((equal (car lc) nil) (setq liste lt))
        ((equal (post (car lc)) poste) (push (car lc) lt) (tri-post (cdr lc) lt))
        (t (tri-post (cdr lc) lt))
    )
)
```

Moteur d’inférences

Le moteur d’inférences va s’occuper de traiter le prochain fait dans la base de faits.

```
(defvar prochaineRegle (make-rule :name "prochaine-regle" :conditions nil :actions nil))

(defun moteurRun()
    (dolist (regle lr)
        (if (not(equal (rule-name regle) "prochaine-regle"))
            (if (funcall (rule-conditions regle))
                (setf prochaineRegle regle)
            )
        )
    )
    (if (not (eq (rule-name prochaineRegle) "prochaine-regle"))
        (funcall (rule-actions prochaineRegle))
        (error "Aucune regle ne peut etre appliquee")
```

```
    )
)
```

Permet de lancer le moteur d’inférences le bon nombre de fois pour qu’il calcul tout les faits.

```
(defun run (nb)
    (cond
        ((equal nb (length lr)) nil)
        (t (moteurRun) (run (+ nb 1)))
    )
)
```