Michael Goletto
Aurélie Leparmentier

Rapport projet de système expert

Jeu l'awalé

Michael Goletto
Aurélie Leparmentier

Table des matières

Michael Goletto
Aurélie Leparmentier

Introduction :

Ce projet a pour but de réaliser un système expert contre lequel un utilisateur peut jouer à un jeu : l'awalé. L'awalé ou awélé est un jeu de société combinatoire abstrait d'origine africaine considéré comme le jeu d'échecs africain. C'est un jeu de type « compter et capturer » dans lequel on distribue des graines.

Ce jeu comporte un certain nombre de règles qu'il faut respecter à chaque tour du jeu. Parmi les innombrables variantes nous ne nous consacrerons qu'à la plus courante appelée Abapa, reconnue par la fédération internationale.

Tout d'abord, nous ferons ressortir les règles qu'il est important de bien intégrer dans notre développement ainsi que leurs significations d'un point de vue technique.

Puis nous détaillerons l'algorithme conçu pour notre développement.

Nous exposerons les problèmes majeurs auxquels nous avons été confronté et leur résolution.

Et enfin nous vous présenterons le code détaillé.

Michael Goletto
Aurélie Leparmentier

1. Analyse des règles importantes à traiter

1.1. Règles d'initialisation

Ce jeu comporte un certains nombres de règles à appliquer avant de pouvoir commencer à jouer.

Tout d'abord, le plateau de jeu contient 12 trous, 6 par joueur. Dans ces trous sont répartis les 48 graines. Il faut donc à l'initialisation du jeu mettre 4 graines dans chaque trou.

Un sens de rotation est préalablement choisit : le sens inverse des aiguilles d'une montre, ainsi que le joueur qui commence. Pour notre jeu nous avons choisit de toujours faire commencer le joueur qui affronte l'ordinateur, comme dans la plupart des jeux informatiques.

1.2. Règles sur chaque tour

A chaque tour, le joueur décide de vider un trou et égraine toutes les graines de ce trou dans les cases suivantes. On vérifiera que le trou d'arrivé appartient bien à l'adversaire, et qu'il contient deux ou trois graines. Si c'est le cas, ces graines seront ajoutées aux gains du joueur actuel, retirées du nombre de graines que l'adversaire possède et aussi du plateau. Puis, on vérifiera le trou précédent, si lui aussi appartient à l'adversaire et contient deux ou trois graines, on effectuera la même procédure que précédemment, ainsi de suite jusqu'à qu'un des trous ne possède plus deux ou trois graines, ou que le trou n'appartienne plus à l'adversaire.

On doit prendre en compte le fait que si lorsque le joueur égraine, il retombe sur sa case de départ, il ne devra pas la remplir. Pour cela on vérifie si le nombre de graines de graines est compris entre 12 et 23, on ajoute le nombre de graines que contient le trou à l'identifiant du trou. Puis on enlève 12 pour trouver le numéro du trou sur lequel on doit arriver et on ajoute 1 qui correspond au trou initial qu'on a sauté lors du passage. De la même manière si les graines sont comprises entre 24 et 35, mais cette fois si on retire 24 et on ajoute 2, et ainsi de suite jusqu'à 48.

Mais également, on devra vérifier le fait que l'adversaire doive toujours avoir des graines dans son camp et dans le cas contraire, on doit lui en fournir. Pour vérifier cette règle dans le traitement, on regarde si le slot sesGraines de l'adversaire est égal à zéro, puis on calcule les coups qui peuvent être joué pour le nourrir. Les coups pouvant être joués sont les trous qui possèdent un nombre suffisant de graines pour arriver jusqu'au camp adverse. Donc le premier trou doit contenir au moins 6 graines, le deuxième au moins 5 et ainsi de suite.

Enfin, on doit faire attention au fait que si un coup doit priver l'adversaire des toutes ses graines, ce coup est joué mais aucune graine n'est collectée. Cette règle est contrôlée en regardant si tous les trous de l'adversaire comporte 0, 2 ou 3 graines et dans ce cas si le trou d'arrivée du coup est le dernier trou du camp adverse.

Après chaque tour des vérifications doivent également être faites afin d'assurer la cohérence du jeu.

1.3. Entre chaque tour

Après chaque tour, soit de l'ordinateur, soit du joueur, on exécutera des règles de gestion du jeu :

- une vérification du nombre de graines gagnées par chaque joueur, si celle-ci est égale ou supérieur à 25, le joueur en question gagne la partie.
- si dans le jeu il reste moins que six graines, le joueur possédant le plus de graines gagne la manche.

À partir de ces règles et de leurs contraintes techniques, nous avons constitué un algorithme.

2. Construction de l'algorithme

2.1. Choix des templates

Pour les templates, nous avons choisi de modéliser un plateau de jeu, des joueurs, ainsi que les trous.
La modélisation du plateau de jeu nous a paru indispensable pour pouvoir afficher l'état du jeu tout au long de la partie. Le plateau contient :

- nbGraines : qui représente le nombre de graines total qui se trouvent sur le plateau de jeu. Il est utilisé pour pouvoir arrêter la partie si il ne reste que 6 graines sur le plateau.
- lesTrous : ce sont tous les trous du jeu
- nbTours : permet de gérer les tours pendant le jeu.

La modélisation des joueurs est obligatoire pour l'affichage en cours de partie et pour pouvoir distinguer le joueur de l'ordinateur. Le template joueur contient :

- id : l'identifiant du joueur. Comme il n'y a que deux joueur il est égal à 1 ou 2. L'ordinateur aura toujours l'identifiant 2.
- Nom : le nom des joueurs. Par défaut, il prend la valeur « ordinateur », de ce fait lors de l'initialisation du jeu, il est inutile de modifier ce slot.
- sesGains : il représente les graines que le joueur a capturé tout au long de la partie. Il est utilisé pour arrêter le jeu si un des joueurs a capturé au moins 25 graines.
- sesGraines : représente les graines que le joueur possède dans sa partie du jeu. Il est utile dans le cas où le joueur n'a plus de graines dans sa partie du jeu et que l'adversaire doit le nourrir.
- sesTrous : c'est la liste des trous du joueur. Par convention, on établi que le joueur 1 possède les trous de 1 à 6 et que l'ordinateur possède les trous de 7 à 12.
- Joue : il indique si le joueur a joué ou non sur le tour en cours.
- Init : donne la valeur du premier trou du joueur.

La modélisation des trous est utile pour l'initialisation du jeu, mais surtout pour connaître le nombre de graines présentes dans chaque trou. Il contient :

- idTrou : c'est l'identifiant du trou. Il va de 1 à 12.
- Contenu : il représente le nombre de graine qu'il y a dans le trou.

Ces templates ont été conçus pour pouvoir gérer au mieux les contraintes de ce jeu.

2.2. Algorithme humain

Nous avons décidé de séparer les traitements qui sont fait pour le joueur « humain », que nous appellerons joueur 1 par la suite, de ceux réalisés pour l'ordinateur.

A chaque tour où le joueur 1 joue, on effectue le traitement suivant :

```
SI      (tous les trous de l'adversaire sont vides)
ALORS
    SI  (il n'existe pas de coup pouvant nourrir l'adversaire)
    ALORS       victoire du joueur actuel
    SINON       on cherche les trous qui peuvent être joué et le joueur a obligation de
                jouer un de ces trous
    FINSI
SINON renvoie la liste des coups comportant l'ensemble des trous du joueur
FINSI

POUR (les trous numéroté de 1 à 6)
FAIRE
        SI (le trou possède au moins une graine) ET (les trous appartiennent à la liste des
coups obligatoires)
        ALORS
                Ajouter le trou à la liste des coups possibles
        FINSI
FINPOUR

FAIRE
Récupération du trou voulant être joué
PENDANT QUE (le trou n'appartient pas à la liste des coups possibles)


SI      (le trou a moins de 12 graines)
ALORS       répartition normale des graines à jouer
SINON       répartition des graines en évitant le trou de départ lors du passage dessus
FINSI

SI      (le joueur adverse n'a plus de graines après le coup)
ALORS       le coup est joué mais aucune capture n'est faite
SINON
        PENDANT QUE     (le dernier trou* du joueur adverse possède à la fin 2 ou 3
graines)
        FAIRE récupération des graines
```

```
            La variable le dernier trou devant le trou précédent
        FINPENDANT QUE
FINSI
```

Le slot joue du joueur passe alors a oui pour le joueur 1 et à non pour l'ordinateur.

2.3. Algorithme ordinateur

L'algorithme de l'ordinateur reprend principalement celui du joueur, excepté pour le choix du trou à vider. Celui-ci est fait par une intelligence artificielle.

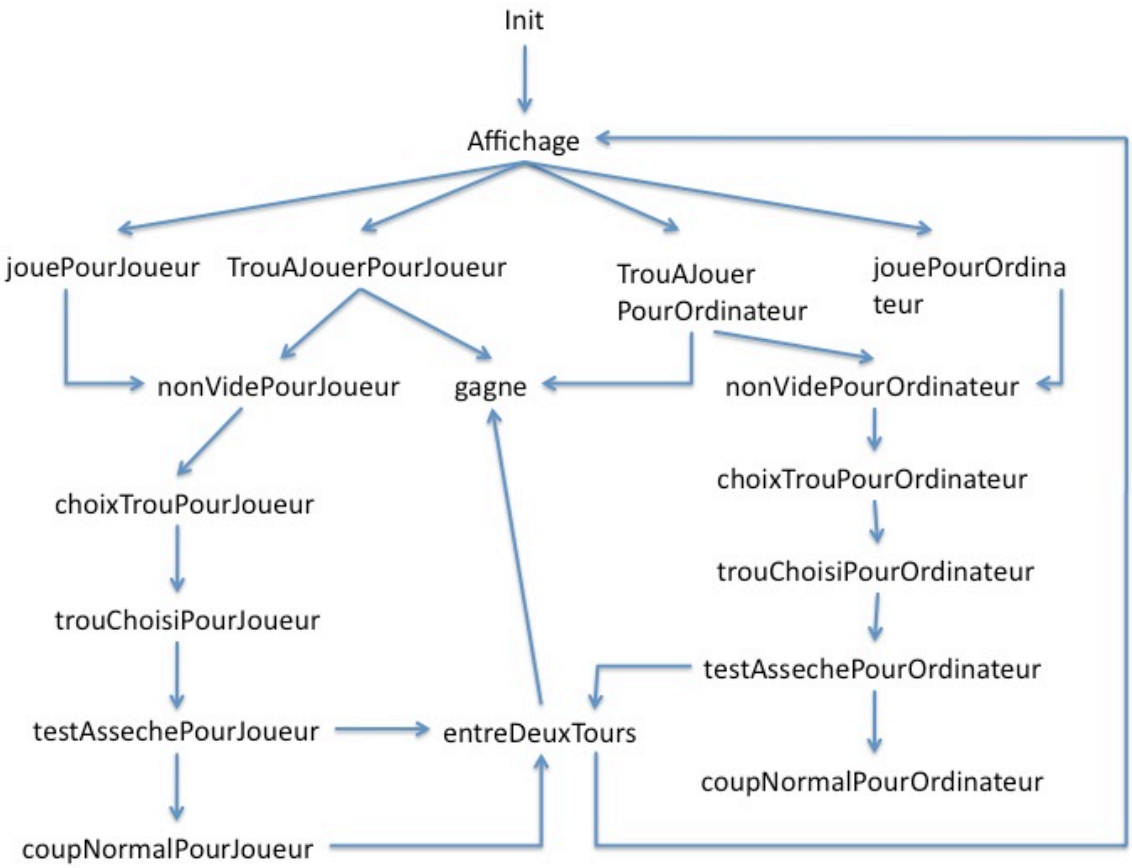
Différentes solutions d'intelligence artificielle ont été envisagées telles que :

- Aléatoire : sélection aléatoire du trou.
- Simple : redonne le maximum de graines à l'adversaire.
- Simple évoluée offensive : vérifie les coups possibles, et effectue celui qui met le plus trous de l'adversaire à deux ou trois graines.
- Simple évoluée défensive : vérifie les coups possibles et effectue où le moins de ses trous ont deux ou trois graines.
- Moyenne : fonction de coût intégrant les possibles coup adverse
- Quasi parfaite : génère 889 milliards de position possible.

Cependant, nous avons opté pour une intelligence simple évoluée défensive, celle-ci ne demandait pas d'ajouter un nouveau template, ni même de récupérer le nombre de graines par trou de l'adversaire. Afin d'optimiser au maximum l'intelligence artificielle, nous avons implanté, pour le premier trou, la sélection du dernier trou : celui-ci s'avère, dans tous les cas possible, le meilleur coup d'ouverture.

```
SI (premier tour)
ALORS jouer le trou numéro 12
SINON
        POUR (les trous de 7 à 12)
        FAIRE
                Inscrire dans une liste le nombre de graine par trou après répartition
        FINPOUR
        MeilleureListe <- listeCoup1
        POUR (les listes générées)
        FAIRE
                SI (MeilleurListe à moins de trous ayant 2 ou 3 graines que ListeItération)
                ALORS
                        MeilleurListe <- ListeItération
                FINSI
        FINPOUR
FINSI
```

2.4. Cycle de vie d'une partie en fonction des étapes



3. Problèmes rencontrés

Lors de la réalisation de notre projet, nous avons rencontré des problèmes qui nous ont amené à repenser l'algorithme.

Tout d'abord, nous avons du gérer le fait que des instructions de ce type (trou (idTrou 1) (contenu ?c)) ne peuvent pas être placées dans la partie action d'une règle. De ce fait pour rechercher dans tous les trous de l'adversaire le contenu de ses trous, on ne peut pas l'utiliser dans une boucle. Il a donc fallu séparer le traitement en plusieurs règles et récupérer le contenu de tous les trous.

Un autre problème s'est alors posé, il a fallu trouver un moyen d'enchainer les règles dans un ordre précis puisqu'on avait séparé de grandes règles en plusieurs petite. Pour cela on a créé un fait étape qui retrace les différentes étapes du traitement.

Nous avons également créé des variables globales pour pouvoir gérer des données qui sont utilisées dans plusieurs règles, comme par exemple la liste des trou qui doivent être joué pou nourrir l'adversaire s'il n'a plus de graines.

Conclusion :

Ce projet nous a permis de mettre en application tous ce que nous avons appris pendant ce semestre en introduction aux systèmes expert. Il a été bénéfique dans la mesure où nous mieux compris la logique de programmation et surtout l'utilité de définir des priorités et des étapes dans nos règles. En effet, sans l'utilisation des étapes, nous n'aurions pas pu créer un enchaînement précis des règles.

De plus nous avons été confronté à des problèmes que nous n'avions pas eu lors des travaux pratiques et nous avons donc dû approfondir nos connaissances dans le langage CLIPS.

Le développement d'un système expert est une tâche qui nécessite beaucoup d'organisation, dont nous avons su faire preuve en commençant notre raisonnement par un algorithme au lieu de commencer à coder sans poser au préalable les problèmes important de ce projet.

En conclusion, ce projet nous a permis d'élargissement nos connaissances informatiques ce qui sera bénéfique pour le stage de fin d'année.