RAPPORT DE PROJET PROLOG

Jeu des 7 familles

Lucille Lecoutre – Tristan Roche

Introduction

Nous avons souhaité dans le cadre de ce projet développer une intelligence artificielle capable de jouer au jeu des sept familles avec un joueur humain. Afin d'éviter toute ambiguïté, voici un rappel des règles du jeu :

On dispose d'un jeu de sept familles de six cartes chacune. Le but est de collectionner toutes les cartes d'une famille en les demandant au joueur adverse et ce pour le plus de famille possibles.

Départ du jeu : on distribue 7 cartes à chaque joueur, le reste des cartes constitue la pioche.

Tour de jeu : un premier joueur demande à un deuxième joueur une carte qu'il souhaite acquérir. On n'a pas le droit de demander une carte dans une famille dont on ne possède pas au moins une carte.

- Si le joueur adverse possède la carte, il la donne au joueur qui la lui a demandée qui a le droit de rejouer.

- Si le joueur adverse ne dispose pas de la carte, le joueur qui demande prend la première carte de la pioche. S'il pioche la carte qu'il vient de demander, il a le droit de rejouer (et de hurler "Bonne pioche" par la même occasion). Si ce n'est pas la carte qu'il a demandé, le tour s'arrête et c'est à l'autre joueur de lui demander une carte. Lorsqu'un joueur réussi à collecter toutes les cartes (6 au total) d'une même famille, il les sort de son jeu.

Fin du jeu : lorsque toutes les familles ont été constituées, celui qui en a le plus gagne la partie.

Configurations particulières : si un joueur n'a plus de cartes, il prend la première carte de la pioche pour reconstituer son jeu. Si les deux joueurs se retrouvent dans cette situation au même tour (ce qui peut arriver si l'un des joueurs n'a qu'une carte qui complète la famille de l'autre joueur) celui qui a demandé une carte se sert le premier dans la pioche.

Nos objectifs initiaux étaient en trois temps :

1) implémenter le moteur jeu pour permettre de jouer une partie complète.

2) ajouter des stratégies de base à l'IA, comme par exemple demander des cartes dans les familles pour lesquelles il dispose du plus de cartes.

3) améliorer les stratégies de l'IA en prenant en compte le comportement du joueur humain (représentation du jeu de l'adversaire).

Au cours des TPs, nous nous sommes rendu compte que nos objectifs étaient trop ambitieux, nous avons donc décidé de nous concentrer sur le moteur de jeux, dans le but que ce qui a été commencé fonctionne correctement.

Une intelligence artificielle sommaire a été créée afin de pouvoir tester une partie complète. Elle se contente de choisir une carte au hasard parmi les cartes qu'elle peut demander.

Notice d'utilisation

Charger le fichier nommé *projet.pl* et lancer le jeu à l'aide du prédicat *main* (qui ne prend pas d'argument).

Les cartes sont distribuées et votre jeu s'affiche sur la console. Vous commencez de jouer. Pour demander une carte, il faut donner d'abord la famille dans laquelle on souhaite demander une carte puis le rang de la carte.

Les familles possibles sont les suivantes : a, b, c, d, e, f, g.

Les rangs possibles sont les suivants : 1, 2, 3, 4, 5, 6.

(Initialement, les noms étaient plus long, mais afin d'éviter les erreurs de saisie, ils ont été réduits à de simples lettres)

Il n'est pas possible de demander une carte qui n'existe pas dans le jeu, ni de demander une carte dans une famille pour laquelle vous ne possédez pas au moins une carte. Si vous le faites, Prolog vous redemandera la carte que vous souhaitez lui soutirer. Tant que vous récupérez la carte demandée, c'est à vous de rejouer. Le prompt s'affiche à nouveau et les cartes piochées (ou récupérées à l'adversaire) sont automatiquement intégrées dans la main qui s'affiche à l'écran.

Le tour de jeu de l'intelligence artificielle est géré automatiquement, la carte demandée est affiché et retiré automatiquement si présente dans votre jeu, un affichage informe du résultat de l'action de l'ordinateur. Dans le cas où l'ordinateur fait une famille, vous êtes également informé.

Les tours se succèdent ainsi jusqu'à ce que toutes les familles soient constituées. S'affiche alors le score (nombre de familles réalisées par chaque joueur).

Exemple de session

Demande d'une carte que l'adversaire possède + résultat suivit d'une demande de carte que l'adversaire ne possède pas :

```
--------------------
------- VOUS -------
--------------------

Votre Jeux :
b
| 1
c
| 6
e
| 6
f
| 2
| 4
g
| 4
| 6
Dans la famille : b.
je demande : 2.


[VOUS] : Dans la famille b, Je demande le 2


tiens ! :'(


Votre Jeux :
b
| 1
| 2
c
| 6
e
| 6
f
| 2
| 4
g
| 4
| 6
Dans la famille : b.
je demande : 3.


[VOUS] : Dans la famille b, Je demande le 3

Pioche !
Mauvaise pioche !
```

Tour de jeu de l'IA (carte non possédé) :

```
--------------------
-------  IA  -------
--------------------


[IA] : Dans la famille b, Je demande le 3

Pioche !
Mauvaise pioche !
```

Tour de jeu de l'IA (2 x carte possédée + 1 fois carte non possédé) :

```
--------------------
-------  IA  -------
--------------------


[IA] : Dans la famille e, Je demande le 6


tiens ! :'(



[IA] : Dans la famille b, Je demande le 2


tiens ! :'(



[IA] : Dans la famille f, Je demande le 1

Pioche !
Mauvaise pioche !
```

Exemple de bonne pioche (le jeu d'avant et après ne sont pas montrés) :

```
Votre Jeux :
a
| 1
| 2
| 3
| 4
d
| 4
Dans la famille : a.
je demande : 5.


[VOUS] : Dans la famille a, Je demande le 5

Pioche !
Carte Piochée :
a
| 5

Bonne pioche !
```

Exemple de fin de jeu (famille + fin du jeu) :

```
Votre Jeux :
b
| 2
| 3
| 4
| 5
| 6
Dans la famille : b.
je demande : 1.


[VOUS] : Dans la famille b, Je demande le 1


tiens ! :'(




FAMILLLLLLLLLE !!


--------------------
-------  IA  -------
--------------------
Bravo, vous avez gagné !! (score : 5 à 2)
true.
```

Une session entière n'a pas été montré, un jeu basique comptant généralement plus de 20 tours, nous en avons ici montré 5 de différents types.

Format des données (forme des faits)

Faits statiques

Les faits définissant le jeu de cartes sont déclarés dans le fichier *definitions.pl*.

Les cartes :

Les cartes sont caractérisées par deux composantes : sa famille et son rang. On déclare donc 7 familles et 6 rangs. Le jeu est généré automatiquement en début de partie à partir de ces faits de base, afin d'éviter de déclarer systématiquement toutes les cartes.

La pioche :

La pioche est définie comme une liste de carte. On vérifie par la même occasion que toutes les cartes de la pioche sont différentes. Dans la génération du jeu (prédicat **genererPioche/1** - fichier *moteurJeux.pl*), il n'est normalement pas possible de générer une pioche contenant deux fois la même carte, mais on effectue toutefois une vérification supplémentaire. La pioche étant une liste, on peut lui appliquer des prédicats associés à ce type d'objet.

Faits dynamiques

Jeux et points :

Les jeux (les cartes présentes dans la main de chaque joueur) ainsi que les points sont

amenés à être modifiés au cours de la partie. Ils sont donc déclarés en faits dynamiques. On enregistrera les jeux, la pioche et les points entre chaque tour de jeu.

Fonctionnement global

Organisation du code

Le programme est réparti dans six fichiers selon le rôle des prédicats qu'ils contiennent. :

- *projet.pl* : on définit ici tous les liens entre fichiers, ainsi que la fonction principale lançant le jeu.
- *definition.pl* : ce fichier contient les définitions de base nécessaires au déroulement du jeu.
- *utilitaire.pl* : contient des prédicats généraux, qui ne sont pas spécifiques au jeu.
- *moteurJeux.pl* : prédicats nécessaires au bon fonctionnement (gestion et contrôles) du jeu : distribution, génération, retirer ou ajouter des cartes au jeu, réarmer, vérification de l'existence des cartes, etc.
- *actionsJeux.pl* : prédicats relatifs au déroulement du jeu, notamment définition du tour de jeu humain et de l'IA
- *Interactions.pl* : prédicats relatifs à l'interaction avec l'utilisateur : principalement affichage (affichage de la main, des points à la fin de la partie) et saisie par l'utilisateur (pour demander une carte à l'IA) demande des cartes.

On entre dans le jeu par le prédicat **main/0** sensé être toujours vrai.

Initialisation

On commence par générer le jeu complet et le distribuer avec le prédicat **genererJeux/3** (fichier *moteurJeux.pl*) qui fait appel aux prédicats suivants :

- **genererPioche/1** qui retourne la pioche, i.e. la liste complète des cartes mélangée (prédicat **shuffle/2** dans le fichier *utilitaire.pl*), et qui vérifie si chacune des cartes est présente de manière unique.

- **distribuer/5** : ce prédicat appelé récursivement distribue une carte à chacun des joueurs à chaque fois qu'il est appelé, il s'agit d'un boucle qui tourne 7 fois.

Le prédicat **genererJeux/3** retourne la pioche restante, le jeu du joueur et celui de l'IA, que l'on enregistre à l'aide du prédicat **asserta/1**. Les jeux, la pioche et les points de chaque joueur (initialisés à zéro) seront retirés et réinsérés à chaque fois que l'on aura besoin de les modifier.

Déroulement du jeu

Le programme entre ensuite dans le prédicat jouer/3 (fichier *actionJeux.pl*). Ce prédicat utilise un repeat sur les prédicats **tourJeuxHumain/3** et **tourJeuxIA/3** qui ne ratent jamais, puis sur **controlFinJeux/0** qui n'aboutit que lorsque le jeu des deux joueurs et la

pioche son vides.

Tour de jeu du joueur humain

Le prédicat **tourJeuxHumain/3** est simplement un appel au prédicat **tourJeuxHumain/4** avec pour initialisation le fait que l'appel doit générer un tour.

tourJeuxHumain/4 :

- On teste d'abord si les jeux et la pioche sont vides, cas dans lequel toutes les familles ont été posées sur la table, auquel cas on retournera directement au tour de jeu de l'IA et à **jouer/3** (présence de coupe-choix) où le prédicat **controlFinJeux/0** aboutira.

- Si les jeux et la pioche ne sont pas vides, on teste ensuite si l'appel doit générer un tour.

  - si non, le prédicat est fini, et toutes les couches sont remontées jusqu'au prédicat jouer/3 auquel cas on passera au tour de jeu de l'IA sans essayer d'autres options (présence d'un coupe choix).
  - Si oui, le tour se déroule normalement comme suit.

On initialise le tour avec **initTour/3** qui a pour rôle de remettre une carte dans les jeux vides (appel de **rearmJeux/3** qui permet de piocher une carte dans la pioche, ou dans la main de l'adversaire si la pioche est vide).

On demande ensuite la carte que le joueur veut demander avec **demanderCarteValide/2** : ce prédicat demande, dans une boucle n'aboutissant que si la demande est valide (la carte existe, le joueur a le droit de la demander), la carte que le joueur veut.

La carte est ensuite demandée à l'adversaire grâce à **demander/5** qui fait appel à **demander/8** (cette organisation permettant une meilleure lisibilité du code principal).

Le joueur a demandé une carte, ce qui aura quatre résultats possibles, testés successivement dans l'ordre suivant. Les trois premières options possèdent un coupe-choix.

(i) L'adversaire possède la carte : on la retire de son jeu et on l'ajoute au jeu du joueur actuel (prédicat **volerCarteAdversaire/3**) qui a alors le droit de rejouer. Ce prédicat n'aboutit que si l'adversaire possède la carte demandée. Sinon, on examinera les autres possibilités.

(ii) L'adversaire ne possède pas la carte demandée mais celle-ci est placée sur le dessus de la pioche. On ajoute la carte au jeu et comme il s'agit d'une bonne pioche, on a le droit de rejouer. La présence d'un coupe-choix ici également permet d'éviter d'essayer les autres options et de retourner directement au tour de jeu.

(iii) L'adversaire ne possède pas la carte demandée, on prend la première carte de la pioche. Cette solution sera testée uniquement dans le cas d'une mauvaise pioche grâce au coupe-choix précédent. Le joueur actuel n'a plus le droit de rejouer (dernière variable à false).

(iv) L'adversaire ne possède pas la carte demandée mais la pioche est vide. Cette configuration ne devrait normalement pas arriver (s'il n'y a plus de cartes dans la pioche, la carte demandée est nécessairement dans le jeu de l'IA), mais on n'est jamais à l'abri d'une erreur et une vérification supplémentaire ne peut pas faire de mal.

- **viderFamille/1** : on teste si une famille complète est présente dans le jeu du joueur à l'aide du prédicat **viderFamille/2**. Si c'est le cas, on sort la famille du jeu et on incrémente le nombre de points. Si ce n'est pas le cas, on ne fait rien mais le prédicat reste vrai.

On refait enfin appel à **tourJeuxHumain/4** avec pour dernier paramètre la variable Rejouer qui est vraie si le joueur doit rejouer, faux sinon.

Le tour de jeu de la machine est géré de la même façon que le tour de jeu de l'humain à un prédicat près : le choix de la carte à demander. Dans le tour de jeux humain, cette étape est faite par **demanderCarteValide/2**. Ici, cette action est faite par **choisirCarteIA/2** qui cherche la liste des cartes qu'elle peut appeler puis en demande une au hasard.

Conclusion

Nous avons finalement réussi à développer un moteur de jeu fonctionnel, il est possible de mener une partie entière contre l'IA. Cela s'est révélé assez complexe et nous a demandé beaucoup plus de temps que prévu. Nous avons été bloqués par l'apparition de certains bugs (dans la gestion des repeat), dont l'origine reste encore un mystère. Ces problèmes ont été rencontrés uniquement sur l'ordinateur de Tristan Roche, et il est probable qu'ils ne soient pas liés à des problèmes d'implémentation Prolog puisqu'ils n'ont pas été rencontrés sur les ordinateurs de l'école. Il est possible que cela soit lié à la configuration de l'ordinateur (l'ordinateur de Tristan étant sous Linux).

Ce problème inexpliqué s'est ajouté à des problèmes rencontrés tout au long du développement à cause de notre manque d'habitude à coder en Prolog, et donc notre manque de connaissance du langage.

Nous avons donc été ralentis et limités dans le développement des stratégies de l'intelligence artificielle que nous aurions souhaité pousser plus en avant. La stratégie de l'IA est pour l'instant assez sommaire : elle se contente de choisir une carte au hasard parmi celles qu'elle peut demander. On peut noter cependant pour sa défense qu'il lui arrive parfois de compléter des familles.

Les objectifs initiaux que nous n'avons pas atteints étaient les suivants : d'abord implémenter une stratégie simple un peu plus élaborée, puis mettre en place une représentation du jeu de l'adversaire. Pour la première étape, nous prévoyions d'implémenter une autre stratégie dans le choix des cartes : l'IA demanderait des cartes dans les familles dans lesquelles il dispose du plus de cartes dans son jeu.

Ensuite, on peut remarquer qu'un élément de stratégie typiquement humain aurait été intéressant à ajouter : lorsque l'on s'est fait voler une carte par l'adversaire et qu'il nous reste encore des cartes dans cette famille, un premier réflexe est de redemander à l'adversaire les cartes qu'on vient de se faire voler pour les récupérer. Un joueur humain aura aussi plutôt tendance à essayer de compléter des familles pour lesquelles il a déjà beaucoup de cartes en main. Il serait donc possible pour l'IA, en intégrant ces informations, d'avoir une représentation déjà intéressante du jeu de l'adversaire.