



Rapport final : Hanabi

Université Pierre et Marie Curie
Projet ANDROIDE
2015-2016

Antunes Costa Gonçalves Daniel
Assmann Catalina
Hubert Cédric
Wolfrom Matthieu

Table des matières

1 Présentation du projet	1
1.1 Contexte	1
1.2 Présentation du jeu Hanabi	1
1.3 Objectifs	1
1.3.1 L'application	1
1.3.2 La logique épistémique	2
2 Expression des besoins - A REVOIR ?	2
2.1 Besoins fonctionnels	2
2.1.1 L'interface Graphique	2
2.2 Besoins non fonctionnels	4
2.3 Critères d'acceptabilité du produit	4
3 Conception	5
3.1 Déroulement du projet - EN COURS	5
3.1.1 Organisation du projet	5
3.1.2 Programmation	6
3.2 Architecture - EN COURS	6
3.2.1 Modèle	7
3.2.2 Vu	7
3.2.3 Contrôleur	8
3.3 Diagrammes de classe - A FAIRE	8
4 Logique et stratégie	8
4.1 Rappels de logique	8
4.2 Choix de modélisation	8
4.3 Stratégies envisagées	10
4.3.1 Situations de jeu intéressants - A FAIRE	10
4.3.2 Intelligences artificielles - A FAIRE	10
5 Manuel d'utilisation	13
5.1 Lancer une partie	13
5.2 Jouer une partie	14
5.3 Sauvegarder une partie	15
5.4 Fin de la partie	15
6 Tests - A FAIRE	16
7 Conclusion - A FAIRE	16
7.1 Difficultés rencontrées	16
7.2 Perspectives d'amélioration	16

1 Présentation du projet

1.1 Contexte

Ce projet se déroule dans le contexte de l'UE Projet de première année de Master ANDROIDE de l'UPMC. Le but initial de ce projet était l'utilisation d'une intelligence artificielle qui exploite la logique épistémique afin d'appliquer la meilleure stratégie possible pour le jeu Hanabi.

1.2 Présentation du jeu Hanabi

Hanabi est un jeu de cartes coopératif composé d'un ensemble de cartes et de jetons. Les cartes représentent des feux d'artifice et les joueurs doivent collaborer afin de composer cinq feux d'artifice de couleur différente. On dispose au début de la partie de 8 jetons «indice» qui représentent le nombre d'indices qu'il est encore possible de donner, ainsi que 3 jetons «éclair» qui représentent le nombre d'erreur que l'on peut faire avant de perdre la partie. Les jetons sont retournés lorsqu'un indice est donné ou une erreur commise. Un jeton d'indice peut être remis sur son côté initial lorsqu'on défausse une carte.

Personne ne connaît ses propres cartes, mais chacun voit les cartes des autres. En utilisant les indices et les cartes visibles, les joueurs doivent essayer de poser les feux d'artifice en suites croissantes de cartes de même couleur.

Le score final est la somme des valeurs des dernières cartes posées pour chaque couleur.

Pour réussir ce défi, chaque joueur peut à son tour effectuer plusieurs actions différentes : donner un indice à un autre joueur, défausser une carte ou jouer une carte.

Un indice consiste à indiquer quelles cartes sont d'une certaine couleur ou d'une certaine valeur dans la main d'un joueur. Un indice négatif peut aussi être donné, indiquant qu'un joueur ne possède aucune carte d'une certaine couleur ou valeur. A chaque indice donné, un jeton «indice» est utilisé et donc retourné. Si le joueur choisit de défausser une carte, il regagne un jeton «indice». Dans le cas où un joueur tente sa chance et joue une carte, si celle-ci est valide, elle est ajoutée dans la colonne de sa couleur. Cependant, il est possible que la carte ne complète pas la suite de sa couleur de feu d'artifice, dans ce cas un jeton «éclair» est retourné et la carte en question est mise dans la défausse. De plus, une fois que trois jetons éclairs sont retournés, le jeu est fini et les joueurs ont perdu.

1.3 Objectifs

1.3.1 L'application

L'objectif principal de ce projet est de créer un logiciel qui permet à un joueur humain de jouer à Hanabi avec une ou plusieurs intelligences artificielles, les plus performantes possibles.

Le joueur pourra choisir le nombre de joueurs, entre deux et cinq. Pour une partie entre 2 ou 3 joueurs, chacun a 5 cartes. S'ils sont plus nombreux, chaque joueur n'a que 4 cartes. L'utilisateur peut choisir entre quatre intelligences artificielles différents pour chacun des joueurs virtuels.

L'interface graphique permettra à l'utilisateur de suivre le déroulement de la partie, en visualisant les cartes de son équipe mais pas les siennes. Il pourra voir aussi la défausse et les indices donnés sur chaque carte.

Il y aura des options pour sauvegarder la partie et la continuer ultérieurement.

1.3.2 La logique épistémique

Hanabi étant un jeu basé sur la coopération entre les joueurs pour permettre à chacun de connaître ses cartes et ainsi de les jouer. La logique épistémique est un moyen de représenter et de raisonner sur les connaissances d'un ou plusieurs agents, c'est donc l'approche qui a été retenue pour ce projet.

2 Expression des besoins - A REVOIR ?

2.1 Besoins fonctionnels

Le logiciel doit permettre à l'utilisateur de jouer une partie du jeu Hanabi avec d'autres joueurs artificiels. Pour ceci, nous avons besoin de créer les fonctionnalités suivantes :

2.1.1 L'interface Graphique

Un menu d'accueil

1. Les règles du jeu

L'utilisateur peut cliquer sur ce bouton pour qu'on lui affiche les règles du jeu.

2. Continuer une ancienne partie

L'utilisateur peut cliquer sur ce bouton pour qu'on lui affiche les parties sauvegardées et en choisir une pour la continuer.

3. Commencer une nouvelle partie

En cliquant sur ce bouton, le joueur voit une fenêtre où il doit préciser avec combien d'IAs il veut jouer (maximum 4 IAs). Il doit aussi préciser si les IAs jouent toutes avec la même stratégie ou avec des stratégies différentes.

L'affichage de la partie

1. Visualiser les cartes des autres joueurs

Les cartes des autres joueurs sont visibles par l'utilisateur.

2. Visualiser les jetons indices restants

Les jetons sont affichés sur la table et donc visible par l'utilisateur. Une fois utilisé, un jeton n'est plus visible.

2.1 Besoins fonctionnels

3. Visualiser les jetons éclairs retournés

Les jetons éclairs sont tous affichés sur la table, retournés ou non.

4. Visualiser les cartes jouées

Les cartes jouées sont visibles sur la table.

5. Visualiser les cartes défaussées

En cliquant sur la défausse, l'utilisateur peut revoir les cartes défaussées. L'interface affiche toutes ces cartes, on peut donc voir leurs couleurs et leurs valeurs.

6. Visualiser les indices donnés pour les cartes de chaque joueur

A chaque indice reçu, l'utilisateur le voit affiché sur la carte concernée. Les indices sont soit une couleur, soit une valeur. L'utilisateur peut aussi voir les indices donnés aux autres joueurs, pour connaître leurs connaissances sur leurs cartes.

7. Jouer une carte

L'utilisateur peut cliquer sur ce bouton et ensuite choisir la carte qu'il veut jouer. La carte est donc affichée et est soit posée sur la table, soit défaussée si la carte ne convient pas. Si elle est défaussée, un jeton d'éclair est retourné sur la table.

8. Défausser une carte

L'utilisateur peut cliquer sur ce bouton et après choisir la carte qu'il souhaite défausser. Elle est donc ajoutée à la pile de cartes défaussées, et un jeton d'indice est de nouveau affiché pour pouvoir être réutilisé.

9. Donner un indice

Pour donner un indice, l'utilisateur clique sur ce bouton puis sur la main qui l'intéresse. Il choisit ensuite soit une couleur, soit une valeur, qui est attribuée aux cartes correspondantes. Un jeton d'indice est donc supprimé, puisque utilisé.

10. Sauvegarder la partie

L'utilisateur peut donner un nom à sa partie et la sauvegarder pour pouvoir la continuer à ultérieurement.

11. Fin du jeu

Le résultat final de la partie est communiqué via une annonce. Cette dernière affiche si les joueurs ont gagné ou perdu. S'ils ont perdu, la cause de la défaite est affichée (3 éclairs). S'ils ont gagné, c'est le score de la partie qui est affiché. L'utilisateur peut ensuite choisir s'il veut faire une nouvelle partie, continuer une autre partie ou quitter le jeu.



FIGURE 1 – Prototype d’interface graphique sur lequel on se base pour notre implémentation.

2.2 Besoins non fonctionnels

L’intelligence artificielle doit décider de son prochain coup en un temps limité afin que la partie se déroule normalement. Il est donc indispensable que les IAs soient efficaces et optimisées.

Nous avons réfléchi à plusieurs possibilités pour les IAs :

- **Une IA simple** : cette IA joue d’une manière basique : elle joue une carte valide si possible (quand elle connaît la carte, et elle peut être posée), sinon, elle donne des indices aléatoires tant que des jetons sont disponibles, dans le cas contraire elle défausse une carte (en priorité les cartes déjà posées ou les cartes sans indices).
- **Une IA sans risque** : cette IA ne joue des cartes que quand elle les connaît. Dans le cas contraire, elle évaluera s’il vaut mieux défausser une carte ou de donner un indice et lequel.
- **Une IA qui prend des risques** : cette IA est moins adverse au risque. En utilisant une heuristique, elle évalue les probabilité de réussite des différentes actions possible et choisi le meilleur coup.
- **Une IA omnisciente** : cette IA connaît toutes les cartes (même les siennes), elle se concentre donc sur les priorités entre les différents coups possibles (jouer une carte qu’on connaît, défausser des cartes pour gagner des jetons «indice» ou donner des indices aux autres joueurs)

2.3 Critères d’acceptabilité du produit

Notre but est de simuler un jeu entre deux ou plusieurs joueurs, en utilisant des intelligences artificielles. Nous avons donc besoin d’avoir des IAs efficaces qui ne prennent pas trop de temps pour décider leur prochain coup. Nos critères d’acceptabilité par rapport aux intelligences artificielles sont donc :

-
- De permettre à l'utilisateur de jouer une partie de Hanabi sans devoir attendre trop longtemps pour son tour.
 - L'intelligence artificielle est considérée comme efficace si elle atteint systématiquement un score entre 20 et 25 (le score maximum possible) en ne jouant qu'entre des IAs avec la même stratégie.

Nous voulons aussi que le jeu soit facile et simple à jouer pour le joueur humain, donc une interface claire et compréhensible est indispensable. Il faut que toute personne soit capable de s'en servir. Il serait d'autre part intéressant d'avoir au moins une IA qui joue d'une façon proche d'un joueur humain plutôt que d'essayer de faire un score maximal.

3 Conception

3.1 Déroulement du projet - EN COURS

L'UE Projet se déroule pendant tout le semestre. Nous avons eu quatre mois pour analyser le jeu, compléter notre programme et le tester. N'ayant jamais entendu parler de Hanabi, nous avons du nous renseigner sur le jeu. Pour commencer, nous avons acheté un exemplaire du jeu sur Amazon. Une fois celui-ci livré, nous avons effectué de nombreuses parties afin d'essayer de discerner nos comportements et nos stratégies, déjà dans un premier temps quand nous n'avions pas encore une bonne connaissance du jeu. Nous avons ensuite pu analyser l'évolution de nos choix stratégiques dans le temps.

Dès le début, nous avons eu des idées de méthodes pour avoir un meilleur score, certaines se sont révélées plus efficaces que d'autres. Par exemple, comme la carte de valeur cinq de chaque couleur existe en un seul exemplaire, nous l'avons considérée comme essentielle. Cependant, si l'on jette tous les exemplaires d'une carte de même couleur et de valeur inférieure, nous n'aurons jamais besoin du cinq alors il ne sert à rien de le garder.

3.1.1 Organisation du projet

En attendant l'arrivée du jeu, nous avons découpé les tâches en deux. Il fallait travailler sur l'architecture du code et aussi rédiger le cahier des charges. En séparant notre groupe en deux binômes, nous nous sommes mis au travail.

En écrivant le cahier des charges, nous avons mis au clair les fonctionnements nécessaires pour la conception du jeu et nous a permis de réfléchir aux intelligences artificielles possibles, comme précisé dans **Besoins non fonctionnels**. Nous avons aussi choisi un design pour l'interface graphique.

Une fois l'architecture de base du programme implémentée, plusieurs axes de progrès ont été définis :

- Développement de la première intelligence artificielle
- Développement d'une interface dans le terminal permettant de faire une partie
- Développement de l'interface graphique

Parler des premières classes qu'on a fait ?

Donner la priorité au joueur le plus proche pour des indices

3.1.2 Programmation

Console dummyIA

Il fallait coder le jeu pour pouvoir jouer en console. D'abord nous l'avons codé dans **Partie** et **Partie-Terminal** pour que deux personnes puissent jouer ensemble en affichant les cartes comme une liste. Sur la console, sans une première intelligence artificielle, c'était impossible de jouer proprement à Hanabi du au fait de ne pas pouvoir cacher les cartes des joueurs.

En conséquence, il nous fallait une IA d'essai qui pourrait jouer contre le joueur humain. Nous avons donc programmé un IA simple qui s'appelle **DummyIA**.

Interface Graphique

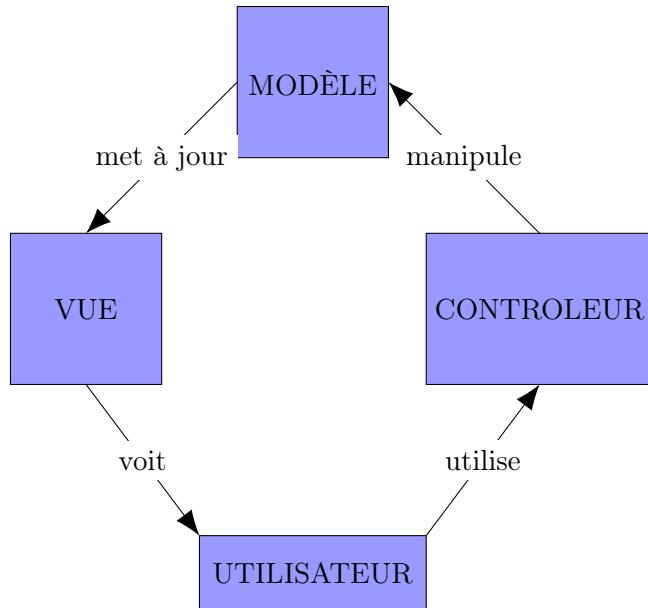
SemiDummyIA

Améliorations IA

Tests et améliorations graphiques

3.2 Architecture - EN COURS

Nous avons utilisé l'architecture MVC (Modèle-vue-contrôleur) pour notre code. Cela consiste à séparer les vues, les classes qui représentent des objets et le contrôleur. Les classes sont des objets, qui sont manipulés dans le contrôleur et qui sont ensuite affichés dans les vues.



3.2.1 Modèle

Dans le modèle nous avons organisé les objets du jeu, comme une main, un joueur et une carte.

La classe principale est ***Partie***. Elle contient toutes les informations nécessaires pour jouer une partie de Hanabi, y compris d'autres classes du modèle :

- ***Joueurs*** est une liste qui contient tous les joueurs de la partie. Un joueur peut être un ***JoueurHumain*** ou un ***JoueurIA***. C'est via cette classe qu'on peut accéder aux mains de chaque joueur grâce à la méthode *getMain()* qui renvoie une ArrayList avec ses cartes. ***JoueurIA*** contient les méthodes de base dont chaque IA a besoin. Par exemple, *coupTrivial* examine les cartes du joueur et permet à l'IA de savoir si une carte de sa main est connue et jouable dans la situation actuelle. En revanche, *carteInutile* regarde si une carte peut être jetée car elle ne servira plus dans cette partie.
- Les variables *jetonEclair* et *jetonIndice* permettent à savoir combien de chaque reste à utiliser dans la partie.
- *dernierTour* est un boolean qui est initialisé à False et change à True une fois que la pile de cartes est vide. Il nous indique qu'après ce tour, la partie est fini. Une fois la partie est fini, le boolean *partieFinie* est mis à True.
- Il y a aussi deux ArrayLists de ***Cartes*** qui sont *pioche* et *defausse*. La première représente les cartes encore dans la pile, et *defausse* les cartes qui ont déjà été jetées par les joueurs. ***Carte*** est sa propre classe et contient sa couleur, sa valeur et si un des deux est connus ou pas par le joueur.

3.2.2 Vu

La vu représente l'interface graphique que voit l'utilisateur. Les deux classes principales sont ***FenetreAccueil*** et ***FenetrePartie***.

Cette première classe est la fenêtre qui s'affiche quand l'application est lancée. Elle permet à l'utilisateur de lancer une partie ou charger une ancienne partie. En cliquant sur nouvelle partie, elle se ferme et une fenêtre de la classe ***Parametres*** s'ouvre. C'est elle qui enregistre les choix de l'utilisateur pour la partie.

La dernière est la classe qui affiche la partie. Son constructeur prend en paramètre la ***Partie p*** et utilise ses méthodes comme *afficherLesJetons()* et *afficherLaDefausse()*, entre autres, pour dessiner la partie pour qu'elle correspond au données de la ***Partie p***. Elle crée une nouvelle ***Table*** qui dessine l'image de la table, mais qui est aussi responsable d'afficher les cartes jouées et la dernière carte défaussée. Puis les mains des joueurs sont affichées grâce au variable *a* qui est de type ***AfficherMains***. Cette dernière s'occupe de dessiner les images des cartes dans les emplacements corrects dépendant du nombre de joueurs.

C'est dans ***FenetrePartie*** qu'on trouve aussi la méthode *saveToFile()* qui permet à l'utilisateur de sauvegarder sa partie comme un FileDialog.

Une fois la partie finie, l'utilisateur verra soit la fenêtre ***PartieGagne***, soit ***PartiePerdu***. Les deux sont des JFrames qui prennent en paramètre une ***Partie p*** et une ***FenetrePartie fp***.

3.2.3 Contrôleur

3.3 Diagrammes de classe - A FAIRE

4 Logique et stratégie

4.1 Rappels de logique

Pour faciliter la compréhension de notre modélisation, voici quelques bases sur la logique épistémique.

On considère la modalité K_i pour chaque agent i. La notation $K_i(\varphi)$ signifie ainsi que l'agent i sait que la proposition φ est vraie.

Les modèles de Kripke sont utilisés pour représenter les mondes possibles, où chaque proposition est donnée comme vraie ou fausse selon la relation définie. Le monde réel est l'un de ces mondes envisageables. Les mondes sont mis en relation pour chaque agent si ce dernier ne peut pas distinguer les mondes concernés à l'aide de ses connaissances.

On note alors :

- Un univers U contenant l'ensemble des mondes du modèle.
- Une relation d'accessibilité \equiv_i pour chaque agent i.
- Une relation $m \vdash \varphi$ qui indique que m vérifie la proposition φ .
- Un modèle est ainsi noté $M = \{U, \{\equiv_1, \dots, \equiv_m\}\}$

Une proposition φ est valide dans un modèle M si $\forall m \in U, m \vdash \varphi$, on le note alors $M \models \varphi$

4.2 Choix de modélisation

Ici le but est d'identifier chaque situation de jeu possible par un monde de Kripke.

La première étape est de modéliser les mondes liés à la main (combinaison de 4 ou 5 cartes sans ordre) du joueur, c'est à dire toutes les mains possibles en prenant en compte les cartes vues, déjà jouées ou déjà défaussées. L'agent correspondant au joueur hésite donc entre tous ces mondes possibles.

Notre structure de Kripke est donc :

$$\begin{aligned} \mu &= \{U, \{R_1\}, I\} \\ U &= \{M_1, \dots, M_m\} \end{aligned}$$

R : A ce stade, R_1 connecte tous les mondes, on suppose que l'on ne génère pas les mondes qu'il sait déjà impossibles.

I : On note $C_{j,c,v}$ la variable affirmant que le joueur j possède une carte de couleur c et de valeur v. Pour n joueurs, un monde M_i contiendra donc toujours un nombre de variables $nv = n*4|n > 3$ ou $nv = n*5|n < 4$

A ce niveau de modélisation, on peut déjà envisager des prises de décisions probabilistes simplement en comptant le nombre de mondes dans lesquels faire telle action aurait un impact positif.

Ensuite, il est également possible pour le joueur de modéliser les mondes entre lesquels il pense qu'un autre joueur hésite. Appelons le joueur qui réfléchit j_1 et celui sur lequel il porte sa réflexion j_2 . Cela pose problème car ce j_2 dispose d'informations que n'a pas j_1 (ses propres cartes), mais on peut tout de même savoir que des mondes sont impossibles grâce à l'information commune aux 2 joueurs (les cartes des autres joueurs, les cartes jouées, les cartes défaussées, les indices donnés). Pour cela, on peut se baser sur les formules de la logique propositionnelle, auxquelles on ajoute la modalité de la connaissance : K_j . Les atomes sont représentés sous la forme : Φ_{j,c,c_1,v_1} tels que "la carte c du joueur j a la couleur c_1 et la valeur v_1 ". L'utilisation de l'axiome de connaissance est donc $K_j(F)$ est vrai dans un monde M si tous les M' successeurs de M pour la relation R_j satisfont F . Ce raisonnement peut être appliqué pour chaque autre joueur.

On a donc maintenant :

$$\mu' = \{U', \{R_1, \dots, R_n\}, I'\}$$

$$U' = \{M_1, \dots, M_{m'}\}$$

Pour les relations R_i , il faut se dire que le joueur i n'hésite qu'entre les mondes où toutes les cartes sont identiques sauf les siennes.

$$\forall i \in [2, \dots, n], \forall (M_j, M_k) \in U' * U',$$

$$R_i(M_j, M_k) \equiv \forall l \in [1, \dots, n] | l \neq i, \forall C_{l,c_1,v_1} \in I(M_j), \forall C_{l,c_2,v_2} \in I(M_k), c_1 = v_1, c_2 = v_2$$

Ensuite, il faut tenir compte des indices donnés par les joueurs qui sont des "annonces publiques" qui viennent enrichir la connaissance commune à tous les joueurs. Grâce à ces annonces et aux formules proposées plus haut, il est possible de supprimer des membres des relations (graphiquement, cela correspond à supprimer des arêtes du graphe), et donc certains mondes deviennent impossibles.

Modélisation carte par carte

Cette fois, l'objectif est de créer des structures de Kripke pour chaque carte plutôt que pour chaque situation. Ainsi, sans information, il y aurait 25 mondes pour chaque carte. On peut ensuite tenter d'appliquer les mêmes raisonnements que dans la modélisation exhaustive.

Univers de la i -ème carte du joueur j :

$$\mu_{i,j} = \{U, \{R_1, \dots, R_n\}, I\}$$

$$U = \{M_1, \dots, M_m\}$$

I : chaque monde contient 2 variables, une pour la couleur et une pour la valeur.

$$\forall M \in U, I(M) = (c, v) | c \in [\text{rouge}, \text{bleu}, \text{vert}, \text{jaune}, \text{blanc}], v \in [1, \dots, 5]$$

4.3 Stratégies envisagées

Cette approche peut être utilisée comme la précédente pour choisir les coups à jouer de manière probabiliste. Cependant, si l'on veut essayer de raisonner sur la connaissance des joueurs, il est nécessaire de mettre en relation tous ces univers, et on en revient donc à la complexité de la modélisation exhaustive.

4.3 Stratégies envisagées

4.3.1 Situations de jeu intéressants - A FAIRE

4.3.2 Intelligences artificielles - A FAIRE

Fonctionnement du DummyIA

Cette IA est la plus simple des IAs. Elle n'est pas très efficace mais elle a servi comme une base sur laquelle baser les suivantes.

DummyIA ne joue une carte que si elle la connaît et que celle-ci est jouable. Si elle n'a pas de carte jouable et qu'il reste des jetons indices, elle donnera un indice aléatoire à un joueur aléatoire.

Sinon, s'il n'y a plus de jeton d'indice, elle va défausser une carte. Les cartes inutiles, c'est-à-dire une carte connue par le joueur et dont un exemplaire identique a déjà été posé, sera prioritaire pour la défausse. Si aucune carte n'est dans ce cas, elle va jeter les cartes sans indices, et en dernier recours, aléatoirement.

Évaluation du DummyIA

Cette IA n'est pas efficace. Elle donne trop d'indices et joue peu de cartes. En plus, les indices ne sont pas très utiles du à leur nature aléatoire. En revanche, elle gère bien la défausse, en donnant priorité aux cartes rendus inutiles en cours de jeu.

Nous avons simulé 50000 parties à 2,3,4 et 5 joueurs entre des DummyIAs, sans utiliser les cartes multicoles. Voici les résultats obtenus :

2 joueurs :	Score max = 13	Score min = 0	Score moyen = 3.3917
3 joueurs :	Score max = 12	Score min = 0	Score moyen = 2.94476
4 joueurs :	Score max = 10	Score min = 0	Score moyen = 2.35198
5 joueurs :	Score max = 9	Score min = 0	Score moyen = 1.96494

Fonctionnement de SemiDummyIA

L'objectif de cette IA était de reprendre les principes de DummyJoueurIA en améliorant les indices donnés. Cependant, nous avons au fur et à mesure de sa conception incorporé des idées supplémentaires(décris dans les paragraphes suivant), ce qui a aboutit à une IA relativement performante, comme l'on peut le constater dans la sous-partie évaluation.

Cette IA suit un algorithme précis.

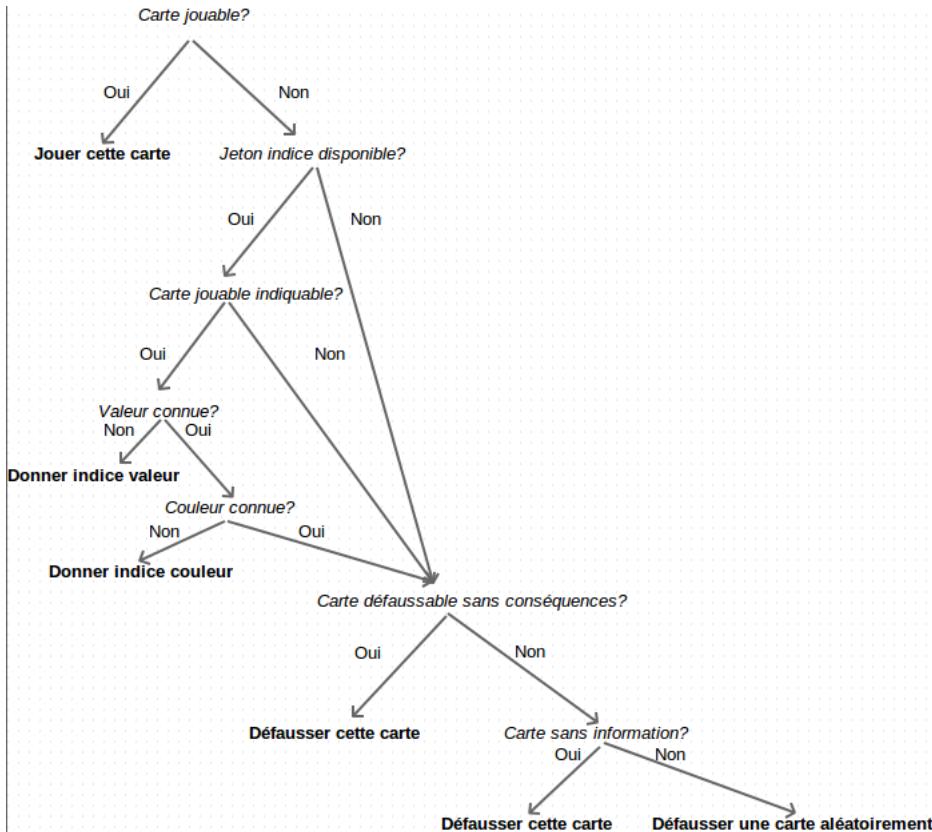
Au début de son tour, si elle a une carte jouable immédiatement, elle la joue.

4.3 Stratégies envisagées

Si ce n'est pas le cas et qu'il reste des jetons d'indice, elle essaie d'en utiliser un. Pour cela, elle parcourt les mains des autres joueurs à la recherche d'une carte jouable. Si elle en trouve une, et que le joueur possédant la carte ne connaît pas sa valeur, l'IA donne cet indice. Si elle connaît la valeur de la carte, mais pas sa couleur, alors l'IA donne l'indice sur la couleur. Après le parcours des mains, si aucun indice n'a été donné (parce qu'aucune carte n'était jouable ou parce que celles jouables étaient déjà complètement connues par leurs propriétaires), alors l'IA donne un indice aléatoire.

Dans le cas où il ne reste plus de jeton d'indice, l'IA va chercher à défausser une carte. Dans ce but, elle regarde dans un premier temps si, parmi ses cartes, elle sait que l'une d'entre elles est "défaussable" (c'est à dire qu'une carte identique a déjà été jouée ou qu'elle ne sera jamais jouable à cause des cartes défaussées jusqu'à maintenant).

Si elle en trouve une, alors elle choisit celle-ci. Dans le cas où l'IA ne trouve pas de carte défaussable, elle va essayer de défausser une carte sur laquelle on ne lui a pas donné d'indices. Il peut arriver qu'aucune carte ne lui soit complètement inconnue, celle à défausser sera alors choisie aléatoirement.



Evaluation de SemiDummyIA

Nous avons simulé 200000 parties à 2,3,4 et 5 joueurs entre des SemiDummyIAs, sans utiliser les cartes

4.3 Stratégies envisagées

multicolores. Voici les résultats obtenus :

2 joueurs :	Score max = 21	Score min = 1	Score moyen = 10.622325
3 joueurs :	Score max = 20	Score min = 6	Score moyen = 14.24381
4 joueurs :	Score max = 19	Score min = 7	Score moyen = 13.88206
5 joueurs :	Score max = 18	Score min = 7	Score moyen = 13.363

On peut constater que les parties comprenant seulement deux joueurs sont les plus difficiles pour cette IA. Il est cependant intéressant de remarquer que c'est pourtant dans cette configuration que la partie au score le plus élevé (21) a été jouée, mais aussi celle au score le plus bas (1). Nous pouvons donc en déduire que le déroulement d'une partie entre deux SemiDummyIA dépend énormément de la disposition des cartes à l'intérieur de la pioche.

Quand le nombre de joueurs augmente, l'écart de score entre la meilleure partie et la pire diminue. C'est logique si l'on comprend qu'avoir plus de joueurs implique d'avoir plus de cartes "en jeu" et non de la pioche, et donc que l'ordre de cette dernière a moins d'importance.

Fonctionnement de HeuristicIA

Le fait que l'IA précédente suive toujours le déroulement "jouer une carte, sinon donner un indice, sinon défausser" apparaissait comme un faible, puisqu'il existe des situations de jeu où il est plus intéressant de donner un indice que de jouer une carte, par exemple. Afin de créer une IA qui prenne de meilleures décisions dans ces cas, nous avons eu l'idée d'utiliser une fonction heuristique qui donne une valeur numérique à un état de la partie. Quand vient le tour de l'IA, cette dernière teste tous les coups qui lui sont possibles, calcule la valeur heuristique pour chaque situation de jeu différente amenée par ces coups, choisit la plus grande, et effectue donc le coup associé. Il faut cependant prendre en compte que les informations du joueur ne sont pas complète, il ne sait pas toujours quelle carte il joue ou défausse, et ne sait évidemment pas ce qu'il va piocher.

Nous avons décidé d'utiliser pour la fonction d'évaluation une combinaison linéaire de sept paramètres :

- Le nombre de points actuel
- Le nombre de jetons indices restant
- Le nombre de cartes jouables immédiatement
- Le nombre de carte défaussables sans conséquence
- Le nombre d'informations (couleur ou valeur) déjà connues sur les cartes jouables
- Le nombre d'informations (couleur ou valeur) déjà connues sur les cartes défaussables
- Le nombre d'informations (couleur ou valeur) déjà connues sur les autres cartes

Évaluation de HeuristicIA

Afin d'obtenir la meilleure fonction possible, il fallait déterminer quel coefficient donner à chaque paramètre (autrement dit, à quoi faut-il donner de l'importance ?). Dans ce but, nous avons utilisé un protocole d'"apprentissage", où le programme tire quasi-aléatoirement le coefficient de chaque paramètre, puis simule 2000 parties avec ces réglages. Si les résultats sont satisfaisant, la configuration est sauvegardée. Ensuite, le programme retire de nouvelles valeurs et recommence. Une fois un nombre suffisant de tirages effectués, nous testons ceux ayant eu les meilleurs résultats sur 50000 parties, afin de pouvoir accorder plus de confiance à ces chiffres.

Ce protocole nous a permis d'obtenir les évaluations suivantes :

2 joueurs :	Score max = 22	Score min = 3	Score moyen = 14.94548
3 joueurs :	Score max = 20	Score min = 8	Score moyen = 15.494
4 joueurs :	Score max = 19	Score min = 7	Score moyen = 14.67022
5 joueurs :	Score max = 18	Score min = 8	Score moyen = 13.33042

Fonctionnement d'EpistemicIA - A FAIRE Daniel

Évaluation d'EpistemicIA

5 Manuel d'utilisation

5.1 Lancer une partie

En lançant le jeu, une fenêtre qui s'appelle ***Bienvenue au jeu Hanabi*** s'ouvre. Dedans s'affiche deux boutons :

- ***Charger Partie*** - L'utilisateur peut choisir une partie existante sur son disque pour la continuer
- ***Nouvelle Partie*** - Lance une fenêtre ***Paramètres de jeu***

Paramètres de jeu propose à l'utilisateur de sélectionner le nombre de joueurs, choisir son pseudonyme et choisir le type d'intelligence artificielle pour chaque joueur. En choisissant Lisa (très bon), qui correspond à EpistemicIA, le joueur peut aussi modifier le niveau de risque de l'IA. 100 signifie qu'elle jouera une carte que si elle a 100% chance d'être jouable.

Il peut aussi cocher la case "Jouer avec les cartes multicolores". Ensuite il peut cliquer sur ***Lancer la partie*** pour commencer.

5.2 Jouer une partie



FIGURE 2 – La partie lancée ressemblera à celle-ci

5.2 Jouer une partie

Une fois dans la fenêtre de la partie, le joueur verra trois boutons :

- **Indice** - Ce bouton permet au joueur de choisir un joueur à lequel il veut donner un indice. Son choix sera affiché par une bordure autour des cartes du joueur choisi. Ensuite il doit choisir soit une couleur, soit un nombre comme indice.
- **Jouer Carte** - Ce bouton permet au joueur de choisir une carte qu'il souhaite jouer. Si son coup est réussi, la carte s'affiche sur la table. Sinon, un jeton éclair se retourne.
- **Défausser** - Ce bouton permet au joueur de choisir une carte qu'il souhaite défausser. Cette carte s'affichera dans la pile de défausse.

Le joueur aura aussi l'option de cliquer sur la pile de défausse pour que les cartes défaussées s'affichent. Quand qu'il veut revenir à la partie, il lui suffit de cliquer en dehors de la zone d'affichage de la défausse.

5.3 Sauvegarder une partie



FIGURE 3 – La défausse d'une partie

5.3 Sauvegarder une partie

Pour sauvegarder une partie, l'utilisateur a deux choix. Soit il peut faire "Ctrl + s", soit il peut aller dans "Fichier" et cliquer sur **Enregistrer Partie**.

5.4 Fin de la partie

A la fin de la partie, une fenêtre s'ouvrira pour déclarer si le joueur à gagné ou perdu. Dans les deux cas, il aura le choix entre trois boutons : **Nouvelle Partie**, **Charger Partie** et **Quitter**.

Si le joueur à gagné, la fenêtre affichera aussi son score final.

6 Tests - A FAIRE

7 Conclusion - A FAIRE

7.1 Difficultés rencontrées

7.2 Perspectives d'amélioration