Projet tuteuré - Hanabi

Lefebvre Florian

June 1, 2023

Contents

1 Remerciements **2**

2 Présentation du projet **2**
2.1 Préambule 2
2.2 Ressources bibliographiques 3
2.3 Règles du jeu 3
2.4 Les choix pour l'implémentation 4

3 Analyse du projet **5**
3.1 L'expression des besoins 5
3.2 Le modèle statique 6
3.3 Le modèle dynamique 7
3.4 L'architecture choisie 9

4 Etat de l'art de la recherche **9**
4.1 Lexique du domaine 9
4.2 Synthèse de l'article "Playing Hanabi Near-Optimally" de Bruno Bouzy [BOU2017] 10
4.3 Synthèse de l'article " The Hanabi Challenge : A New Frontier for AI Research" de Nolan Bard et al [BAR2019] 12
4.4 Synthèse 13
4.5 Notre démarche 14

5 Développement du projet **15**
5.1 Première version en mode console 15
5.2 Deuxième version en mode graphique 18

6 L'algorithme d'IA du Joueur de certitude **20**
6.1 Fonctionnement du joueur de certitude 20
6.2 Implémentation du joueur de certitude 21

7 L'algorithme d'IA de la stratégie de recommandation du chapeau **22**
7.1 Fonctionnement de la stratégie de recommandation du chapeau 22
7.2 Implémentation de la stratégie de recommandation du chapeau 24

8 Nos résultats **27**

9 Conclusions **28**

References **29**

1 Remerciements

Je souhaite adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide pour mener à bien ce projet. Je tiens à remercier sincèrement Monsieur Benjamin Dupont, qui en tant que professeur encadrant, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de mon projet. Ainsi je le remercie pour l'aide et tout le temps qu'il a bien voulu me consacrer. Mes remerciements s'étendent également à l'ensemble des professeurs pour la qualité de leurs cours et tout ce qu'ils m'ont apporté. Enfin je tiens également à remercier mes parents pour le soutien qu'ils m'ont toujours apporté pendant ma scolarité.

2 Présentation du projet

2.1 Préambule

Ce document a été créé dans le cadre du projet tuteuré de la Licence 3 Informatique de Paris 8 pendant l'année scolaire 2022-2023. Ce rapport est un état de l'art concernant le jeu Hanabi et les travaux sur le développement d'Intelligences Artificielles (IA) pour ce jeu. Nous allons parler ici de notre démarche pour résoudre le jeu de Hanabi. Il existe différentes approches pour ce jeu. Nous allons expliquer dans une première partie les règles du jeu de Hanabi. Dans une seconde partie, nous proposons de modéliser avec UML le cahier des charges. Enfin dans une troisième partie, nous présenterons l'état de l'art des différents types d'Intelligence Artificielle s'appliquant à ce jeu.

2.2 Ressources bibliographiques

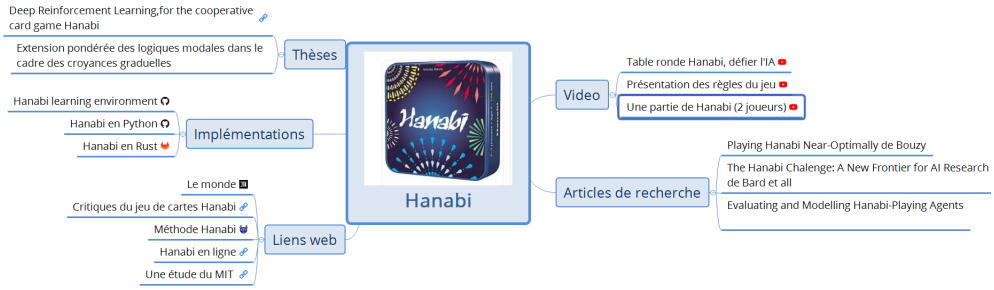

Figure 1: Mind Map de la recherche bibliographique

2.3 Règles du jeu

Figure 2: Les éléments du jeu

C'est un jeu coopératif où les joueurs ne s'affrontent pas mais ce sont des équipes qui atteignent un objectif commun. Le jeu est composé de 5 couleurs (rouge, vert, bleu, jaune, blanc). Il est composé aussi de 8 jetons bleus et 3 jetons rouges. Le but du jeu est de constituer une suite croissante des cartes de 1 à 5 pour chaque couleur. Il faut mélanger les 50 cartes de la pioche. Le jeu est composé au maximum de 5 joueurs. Lorsqu'il y a 2 ou 3 joueurs, chaque joueur reçoit 5 cartes et lorsqu'il y a 4 ou 5 joueurs, chaque joueur reçoit 4 cartes. Le joueur ne doit pas regarder ses propres cartes mais il peut regarder les cartes des autres joueurs. Le joueur doit effectuer l'une des trois actions du jeu. Les trois actions du jeu sont : donner une information, défausser une carte, jouer sa carte.

Donner une information est une action où le joueur doit retirer un jeton bleu du couvercle de la boîte. Il donne une information à un autre joueur sur le chiffre ou la couleur d'un ensemble de cartes de la main de l'autre joueur. Il faut bien évidemment que le joueur qui donne une autre information à l'autre joueur soit complète c'est à dire que l'informateur ne peut pas dire à

l'autre joueur que ce dernier a 2 cartes jaunes alors que ce dernier a 3 cartes jaunes. Exemple : Tu as 2 cartes bleues ici ou tu n'as aucune carte blanche. Exemple 2 : Tu as 2 cartes de valeur 2 ici ou tu as ici une carte de valeur 3.

Une deuxième action est de défausser une carte, cela correspond à jeter une carte qui n'est donc plus utilisable pour constituer les piles. En conséquence, on regagne un jeton bleu. Il pose une carte dans la pile de défausse et il reprends une nouvelle carte et la place dans sa main.

Une troisième action est de poser une carte ce qui signifie que le joueur joue sa carte dans le jeu. La carte que pose le joueur peut compléter le feu d'artifice. Il peut aussi en créer un nouveau dont la couleur n'a pas encore été créée. S'il pose une carte de valeur 5, le joueur peut remettre alors un jeton bleu dans le couvercle de la boîte. Si la carte ne complète aucun feu d'artifice, alors la carte est défaussée et un jeton rouge est ajouté dans le couvercle de la boîte. Si un joueur ajoute une carte de valeur 5 , cela finit un feu d'artifice et il doit ensuite remettre un jeton bleu dans le couvercle de la boîte. Il n'a donc pas a défaussé une carte.

Une partie du jeu Hanabi peut se terminer de 3 façons :

- Si un troisième jeton rouge est mis dans le couvercle de la boîte, alors la partie se termine et cette partie est perdue.

- Si les joueurs ont complété la partie avec 5 feux d'artifices avant que la pioche se termine, alors la partie se termine et c'est une victoire éclatante. Les joueurs ont alors un score de 25 points.

- Si un joueur pioche une dernière carte de la pioche, alors la partie touche à sa faim et donc chaque joueur va jouer une dernière fois y compris celui qui a pioché la dernière carte de la pioche. Lors de ce dernier tour du jeu, les joueurs ne peuvent donc plus piocher de carte. Une fois que le dernier tour du jeu est passé, la partie est enfin terminée et les joueurs peuvent compter leur score.

Pour calculer leur score, ils doivent déterminer la plus grande valeur de chaque couleur de feu d'artifice et font la somme des plus grandes valeurs des couleurs du feu d'artifice. Pour conclure sur les règles de ce jeu, on peut noter qu'il existe des variantes du jeu avec des extensions, mais il ne sera ici question que de la version originale.

2.4 Les choix pour l'implémentation

Nous avons choisi le langage Java comme langage de programmation. Mais tout autre langage doté d'un environnement riche comme Rust ou Python pourrait également être intéressant. Néanmoins, nous choisirons Java car cela nous donne la possibilité d'utiliser un outil de conception UML qui s'appelle Modelio, permettant de faire du forward et du reverse engineering et donc

de pouvoir matérialiser des modèles à partir du code développé. Cet environnement est open source et gère le langage Java. De plus, il est facilement associable à Prolog. Ce qui pourrait être intéressant pour développer un agent d'intelligence artificielle.

3 Analyse du projet

3.1 L'expression des besoins

Le cahier des charges du projet demande de développer le jeu Hanabi avec de l'intelligence artificielle. Le schéma ci-dessous présente les fonctionnalités du jeu et les acteurs autour de ce jeu. L'ordinateur peut faire partie des joueurs et suivra une approche d'intelligence artificielle que nous préciserons par la suite. Les fonctionnalités attendues pour ce jeu sont au départ le paramétrage du jeu et en l'occurence, définir le nombre de joueurs, voir si nous souhaitons utiliser un agent artificiel parmi les joueurs, définir qui jouera le premier, et ensuite lancer le jeu et demander à chaque joueur tour à tour de jouer.

Le fait d'envisager qu'un joueur puisse être un ordinateur est une option du jeu. Elle ne peut être imposée, si les joueurs veulent être complètement libre de leur champs d'action. Néanmoins, cette option est vraiment intéressante si nous voulons donner une orientation originale à ce jeu où l'ordinateur pourrait aider les autres joueurs dans leur objectif de gagner. Des stratégies seront à analyser pour arriver à cette possibilité et leur mise en oeuvre sera déterminante pour évaluer leur pertinence. La fin de ce document présente des travaux de recherche qui vont dans ce sens.

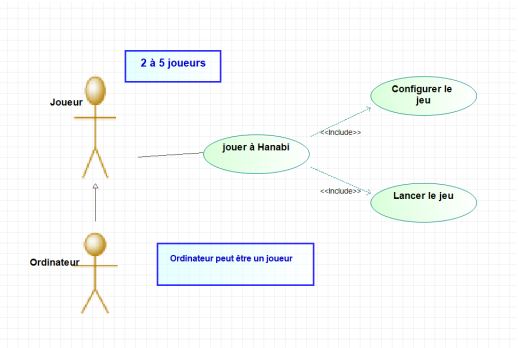

Figure 3: L'expression des besoins

Ci-dessous voici le prototype de l'interface correspondant à ce que nous souhaitons développer lors de l'implémentation du projet. Cette interface n'est qu'une ébauche et évoluera sûrement lors du développement du projet.

De même, les utilisateurs potentiels de cette application, de par leur retour, pourront certainement donner des avis d'améliorations de cette interface qu'il faudra prendre en compte par la suite.

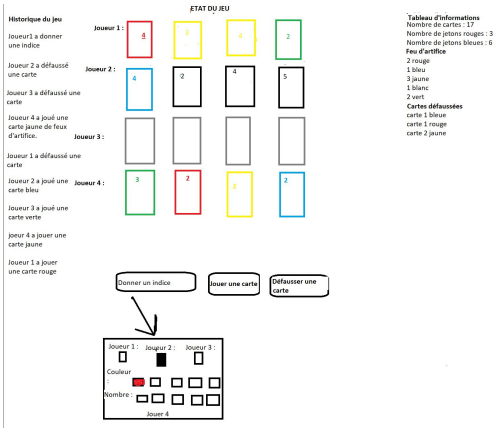

Figure 4: L'interface de notre application

3.2 Le modèle statique

L'approche que nous préconisons pour le projet est d'utiliser la programmation orientée objet. Ci-dessous, voici une première version des classes métiers du cahier des charges du projet. C'est une première version qui ne prend pas en compte les autres classes nécessaires pour le développement de l'application telles que les classes de l'interface Homme-Machine.

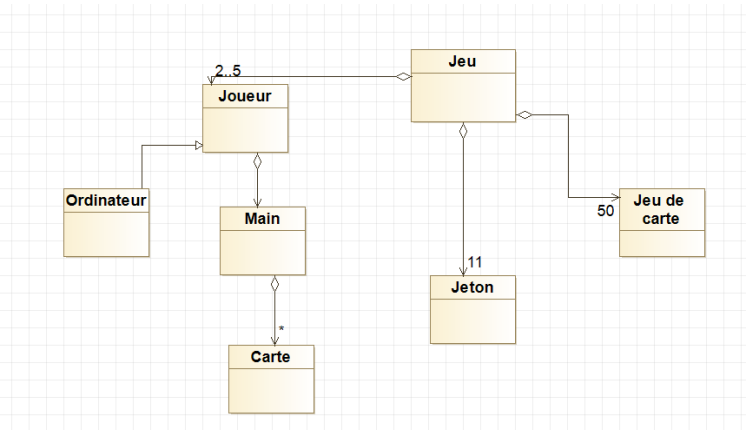

Figure 5: Le diagramme de classe

3.3 Le modèle dynamique

Il est important d'analyser comment les objets évoluent dans le temps et comment ils interagissent entre eux. Les scénarios permettent de matérialiser différentes situations du jeu. Le cas d'utilisation Configurer le jeu se décrit de la façon suivante :

Cas d'utilisation : Configurer le jeu

scénario nominal

1. demander le nombre de joueur (entre 2 à 5)
2. demander le nom des joueurs
3. demander qui commence la partie
4. donner le nombre de carte à chaque joueur

scénario à erreur

1. demander le nombre de joueur (entre 2 à 5)
2. si le nombre de joueur n'est pas correct revenir en 2
3. demander le nom des joueurs
4. demander qui commence la partie
5. donner le nombre de carte à chaque joueur

Le cas d'utilisation Jouer à Hanabi pourrait se traduire par le scénario suivant :

Cas d'utilisation : Jouer à Hanabi

scénario nominal

1. L'objet Jeu est créé
2. L'objet Jeu de cartes est créé
3. Les objets joueur sont créés
4. Le jeu de cartes est mélangé
5. Le jeu de cartes est distribué aux joueurs

6. Le tour du premier joueur commence
7. Il choisit de donner un indice à un autre joueur
8. Le jeu demande au premier joueur de sélectionner le joueur pour recevoir un indice
9. Le jeu demande un type d'indice (couleur ou numéro de carte)
10. Le jeu transmet l'indice à un joueur sélectionné et met à jour sa main pour refléter les nouvelles informations.
11. Le joueur suivant commence
12. Il utilise l'indice pour prendre une décision et choisir de jouer une carte
13. Le jeu vérifie si la carte est valide (par exemple si c'est la carte suivante dans la séquence correcte pour sa couleur, soit il la rejette
14. Le jeu continue ainsi jusqu'à la fin de la partie (victoire ou défaite)

Une partie du jeu évolue dans le temps et se termine pour diverses raisons. Voici ci-dessous le diagramme d'état de l'objet jeu.

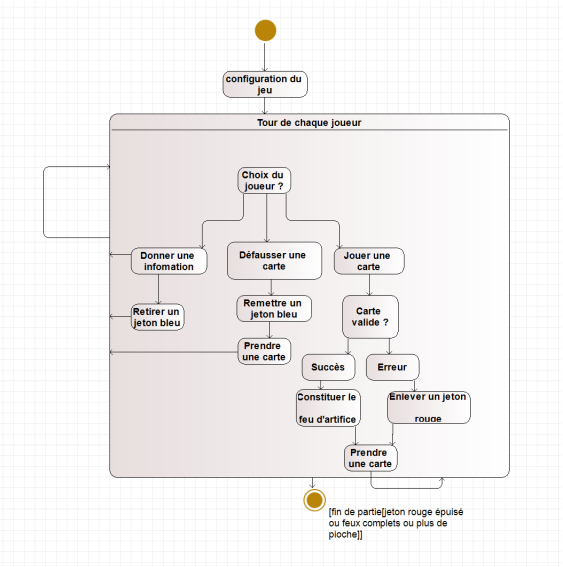

Figure 6: Cycle de vie de l'objet jeu

3.4 L'architecture choisie

L'application, de par la nécessité que chaque joueur ne voit pas ses propres cartes, nécessite une architecture distribuée. Dans un premiers temps, nous envisagerons une version mono-poste pour faciliter les tests. Ensuite, nous nous concentrerons sur la partie IA. En dernier lieu, il serait intéressant d'envisager de répartir l'application sur plusieurs machines. Le schéma suivant montre cette architecture.

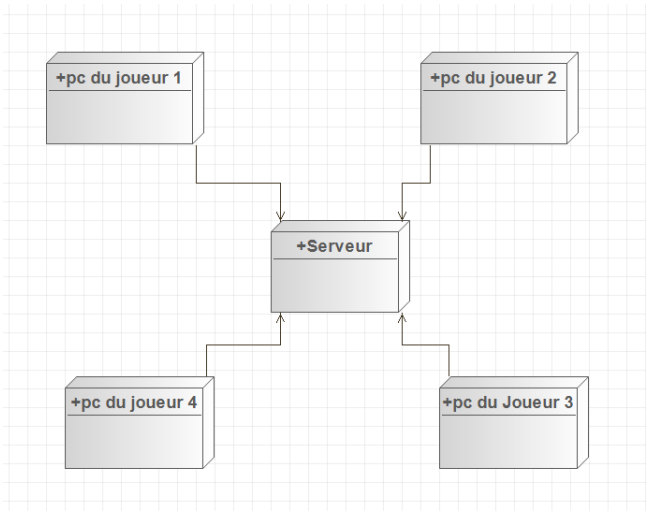

Figure 7: Architecture distribuée de l'application

4 Etat de l'art de la recherche

4.1 Lexique du domaine

Les articles de recherche font souvent appel à un vocabulaire spécialisé dont la définition peut être utile pour comprendre leur utilisation. Je propose dans cette section de les expliquer avant de présenter l'état de l'art de la recherche.

1. **Apprentissage par renforcement**

   L'apprentissage par renforcement (RL pour Reinforcement Learning) fait référence à une classe de problèmes d'apprentissage automatique, dont le but est d'apprendre, à partir d'expériences successives, ce qu'il convient de faire de façon à trouver la meilleure solution. Il consiste à

laisser les ordinateurs apprendre de leurs expériences grâce à un système de récompense ou de pénalité. Il pourrait même s'agir de la clé permettant l'avènement d'une intelligence artificielle générale comparable à celle de l'humain...

2. **Bot**

   C'est une application logicielle programmée pour effectuer certaines tâches. Automatisées, ces applications s'exécutent selon leurs instructions, sans nécessité pour un utilisateur humain de les lancer manuellement à chaque fois. Les bots imitent ou remplacent bien souvent le comportement des utilisateurs humains.

3. **Jeu self-play**

   L'agent joue avec une copie de lui-même

4. **Jeu cross-play**

   L'agent est associé à d'autres types d'agent

5. **Jeu humain**

   L'agent coopère avec un humain

6. **Théorie de l'esprit**

   Le concept de théorie de l'esprit (Theory of Mind en anglais [ToM]) désigne la capacité mentale d'inférer des états mentaux à soi-même et à autrui et de les comprendre.

7. **Ad hoc**

   L'agent joue avec tout type de coéquipier inconnu.

4.2 Synthèse de l'article "Playing Hanabi Near-Optimally" de Bruno Bouzy [BOU2017]

Cet article a pour but de montrer comment bien jouer avec l'ordinateur sur le jeu de Hanabi. Il s'agit de proposer de nouveaux principes pour atteindre de meilleurs résultats sur ce jeu. Il existe comme principe celui dit du chapeau où le score parfait qui a été atteint est 75 % en moyenne pour 5 joueurs et 4 cartes par joueur. Le jeu développé dans cet article s'appelle Hanibal, avec la définition d'un ensemble d'acteurs, le but étant d'obtenir des résultats du jeu quasi optimale du jeu Hanabi. Le principe du chapeau est associé à une recherche en profondeur de l'algorithme.

Je vais expliquer l'idée essentielle qui est le principe du chapeau et ce principe est utile surtout pour bien jouer avec l'ordinateur à ce jeu. Le principe du chapeau est un principe basé sur des mathématiques. Il s'agit de baser ce jeu avec des probabilités. Le chapeau est une valeur à calculer. Il s'agit de représenter une main d'un jouer avec un chapeau ce qui est un nombre h tel que 0 <= h < H . Dans la stratégie de recommandation, H = NCPP * 2. NCPP est le nombre de cartes par joueur. Un joueur du jeu ne voit pas les cartes de ses mains mais voit les cartes des autres joueurs en calculant leur chapeau. Il existe des programmes de recommandations pour calculer cette valeur de chapeau. Le principe du chapeau est puissant car le mouvement d'information informe tous les joueurs et pas un seul joueur ciblé. La stratègie d'information doit informer les autres joueurs sur les valeurs de cartes possible ayant une probabilité élevé.

D'après l'article, l'assouplissement de règles peut donner lieu à des résultats différents. Le premièr assouplissement de règle c'est que les joueurs puissent voir leur propre carte. Un deuxième assouplissement à faire est de pouvoir donner une information ou défausser une carte quelque soit le nombre de jetons bleus.

Les différents joueurs développés dans Hanibal vont maintenant être décrit. Ce sont quasiment tous des simulateurs de bases de connaissances. Les différents joueurs développés dans Hanibal sont :

- **un joueur (lecteur) de certitude**, qui lui utilise la convention suivante: l'information doit être donnée sur les cartes jouables et les cartes inutiles car ces informations sont importantes pour le jeu. Il faut jouer la carte quand celle-ci est jouable avec certitude. Il faut défausser une carte dès que cette carte est jetable avec certitude.

- **un joueur de confiance** : Cela utilise l'idée de la confiance qui est d'informer sur les cartes une fois avant de jouer une carte ou de défausser une carte dans la mesure du possible. Lorsque le joueur donne une information explicite sur le joueur ciblé, le joueur ciblé (joueur informé) doit jouer une carte s'il peut constater lui-même que la carte peut-être jouée en toute confiance. Sinon si le joueur averti (joueur informé) peut remarquer qu'il faut défausser une carte, alors il défausse la carte. Les jetons bleus sont dépensés moins vite.

- **un joueur de recommandation de chapeau** : C'est une stratégie de recommandation qui est identique à la stratégie de Cox. Celle-ci est basée sur le principe du chapeau. La stratégie de Cox[BOU2017] suppose qu'on ne peut pas donner des informations qui ne font pas partis de la main d'un joueur, comme une couleur qui ne ferait pas parti de son jeu . Il s'agit d'une stratégie qui repose sur le principe du chapeau.

- **le lecteur d'information sur le chapeau** : C'est une stratégie

d'information qui repose sur le jeu de probabilités de carte. Cette probabilité de carte est calculée en fonction de l'information publique qui a été donnée dans cette carte. La probabilité dans la main d'un joueur se repose sur la carte visée par la stratégie d'information. Le lecteur d'information sur le chapeau est une simplification de la stratégie de Cox car l'interdiction de donner une information est enlevée dans cette stratégie d'information du lecteur d'information sur le chapeau.

- **Le joueur voyant** est un joueur qui voit ses propres cartes mais pas celles dans la pioche.

- **un lecteur de recherche arborescente** : Ce lecteur est lancé avec une profondeur donnée et avec un certain nombre de distribution de cartes en cherchant à partir d'un noeud aléatoire, tous les autres noeuds générés pour trouver le noeud ayant la meilleure chance.

4.3 Synthèse de l'article " The Hanabi Challenge : A New Frontier for AI Research" de Nolan Bard et al [BAR2019]

Le but de cet article est de voir comment la machine peut prendre des décisions pour jouer à ce jeu de Hanabi.

L'apprentissage automatique a fait de nombreux progrès sur différents domaines et sur des jeux. Cela a stimulé la recherche pour les praticiens de l'intelligence artificielle. Hanabi est un jeu qui élève le raisonnement sur les croyances et sur les intentions des autres agents. Cela correspond à la théorie du raisonnement de l'esprit. Cet article nous dit que développer de nouvelles techniques pour une telle théorie du raisonnement de l'esprit ne sera pas seulement important pour le succès du jeu d'Hanabi, mais aussi dans des efforts de collaborations plus larges, en particulier ceux avec des partenaires humains.

L'interaction multi-agents joue un rôle essentiel dans la vie humaine. Il est souhaitable que les humains puissent communiquer avec d'autre agents dans Hanabi, dans d'autres jeux aussi et également dans des tâches de la vie quotidienne comme par exemple au travail. Il est également souhaitable que les agents artificiels puissent communiquer avec les agents humains et coopérer efficacement. Le comportement idéal d'un agent dépend généralement de la façon dont les autres agents agissent.

Ainsi, pour qu'un agent maximise son coup dans un tel environnement, il doit réfléchir sur la façon dont les autres agissent pour ainsi agir de manière appropriée. Cela donne lieu à la théorie de l'esprit qui est la capacité humaine à imaginer le monde du point de vue d'une autre personne. Hanabi est

en effet défendu dans cet article comme une nouvelle frontière de recherche qui présente le type de défis multi-agents où les humains utilisent la théorie de l'esprit. Un jeu qui est réussi implique qu'il y ait une communication implicite d'informations supplémentaires, ce qui est observable par tous les acteurs du jeu. Les humains jouent aussi régulièrement avec des équipes "ad hoc" qui peuvent avoir des joueurs de différents niveaux de compétence. Cet article propose d'utiliser Hanabi comme problème de référence difficile pour l'Intelligence Artificielle. Il s'agit d'un problème d'apprentissage multi-agents. La façon dont les humains jouent au jeu suggèrent deux défis différents présentés par le jeu Hanabi. Le premier défi est d'apprendre une politique fixe pour tous les joueurs par sa propre façon de jouer. Le deuxième défi est d'apprendre à jouer avec un ensemble de partenaires inconnus avec seulement quelques jeux d'interaction qui est le jeu d'équipe ad hoc.

Il y a ainsi un environnement d'apprentissage de renforcement Hanabi open source qui est appelé Hanabi Environement ayant pour objectif d'améliorer les résultats de recherche.

Hanabi est difficile pour les algorithmes d'apprentissage actuels. Même quand il y a une grande quantité de données et de temps de calcul utilisé, les agents d'apprentissage ont du mal à approcher les performances des règles élaborées dans les jeux à 4 joueurs et ils sont bien en dessous du respect de ses règles pour 3 et 5 joueurs.

Le deuxième défi posée par Hanabi est celui du jeu d'équipe ad hoc. Le but est d'être capable de jouer avec d'autres agents ou même des joueurs humains. Les bonnes stratégies ne sont pas uniques. Un bon joueur doit apprendre à reconnaître l'intention dans les actions des autres agents et s'adapter à un large éventail de stratégies possibles. Cet article propose d'évaluer la performance d'une équipe ad hoc en mesurant la capacité d'un agent à jouer avec un large éventail de coéquipiers inconnus.

Hanabi présente des défis d'apprentissages multi-agents intéressants pour apprendre une politique d'auto-jeu et ainsi une adaptation à une équipe ad hoc. Ces défis sont difficile pour les algorithmes d'apprentissage.

4.4 Synthèse

Plusieurs approches sont possibles pour définir une intelligence articielle pour le jeu Hanabi. Jusqu'à maintenant, les jeux classiques comme les échecs ont été surpassé par des algorithmes d'apprentissage par renforcement. Mais malgré les performances individuelles élevées des agents RL, ils peuvent devenir des coéquipiers frustrants lorsqu'ils sont associés à des joueurs humains, selon une étude réalisée par des chercheurs en IA du MIT Lincoln Laboratory. L'étude, qui impliquait une coopération entre des humains et des agents d'IA

dans le jeu de cartes Hanabi, montre que les joueurs préfèrent les systèmes d'IA classiques et prévisibles basés sur des règles aux systèmes RL complexes [TEC2021].

L'apprentissage par renforcement en profondeur est intéressant dans le sens où il commence par fournir à un agent un ensemble d'actions possibles dans le jeu, un mécanisme pour recevoir les commentaires de l'environnement et un objectif à poursuivre. Puis, à travers de nombreux épisodes de gameplay, l'agent RL passe progressivement de la prise d'actions aléatoires à l'apprentissage de séquences d'actions qui peuvent l'aider à maximiser son objectif.

Ces dernières années, plusieurs équipes de recherche ont exploré une autre voie qui est le développement de robots IA pouvant jouer à Hanabi. Certains de ces agents utilisent une IA symbolique, où les ingénieurs fournissent au préalable les règles de jeu, tandis que d'autres utilisent l'apprentissage par renforcement.

Les systèmes d'IA sont évalués en fonction de leurs performances en self-play, en cross-play et en jeu humain. Pour tester l'efficacité de la coopération homme-IA, les chercheurs ont utilisé SmartBot, le système d'IA basé sur des règles le plus performant en auto-jeu, et Other-Play, un bot Hanabi qui s'est classé le plus haut dans le jeu croisé et le jeu humain parmi les algorithmes basés sur le RL. Selon les sondages des participants de cette étude, les joueurs Hanabi les plus expérimentés avaient une moins bonne expérience avec l'algorithme Other-Play RL par rapport à l'agent SmartBot basé sur des règles.

4.5 Notre démarche

Une approche itérative et incrémentale serait intéressante pour développer cette application. Un prototype rapide de ce jeu peut être réalisé en mode console sans agent intelligent. Une deuxième version pourrait remplacer l'interface console par une interface graphique. Une troisième version serait l'intégration d'un agent intelligent dans notre application. Celui-ci pouvant être développé à base de règle en Prolog. Une étude de la faisabilité d'étudier l'incertitude ou la croyance [LEG2018] avec Prolog pourrait être envisagée. Dans ce projet, on va essayer d'implémenter certains des joueurs de l'article de Bruno Bouzy pour permettre à l'utilisateur de jouer contre différents types d'IA. Enfin, la dernière version pourrait être répartie, voir même être accessible sur internet. Le choix de l'IA n'est pas arrêté mais selon le retour d'expérience des articles de recherche, l'approche d'apprentissage par renforcement, est un echec pour l'instant dans les études réalisées. D'autres voix restent à explorer.

5 Développement du projet

5.1 Première version en mode console

Je vais vous expliquer ici les rôles de chacune des classes que j'ai développé dans mon jeu.

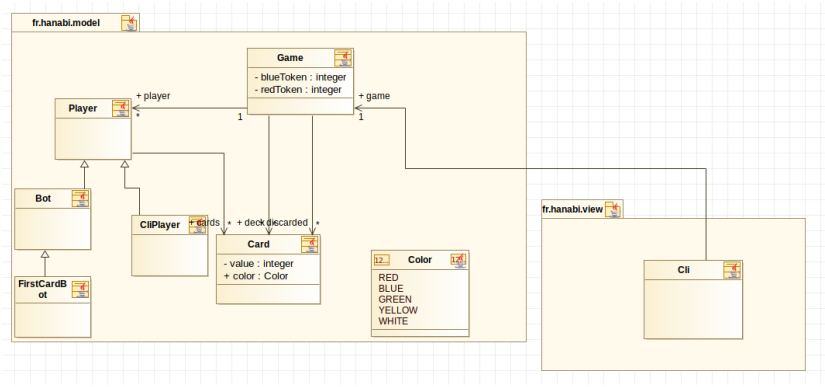

Figure 8: Le diagramme de classes du jeu

La classe Game : c'est la classe principale du jeu et son rôle est donc de gérer toutes les autres classes du jeu. En outre, cette classe permet de générer la pioche du jeu et crée les joueurs. Elle gère ainsi tous les paramètres du jeu tels que le nombre de jetons rouges qui augmentent quand un joueur joue une carte fausse, le nombre de jetons bleus qui est récupéré quand un joueur défausse une carte, etc. Le jeu s'affiche à chaque tour pour chaque joueur et par rapport aux résultats des composants du plateau du jeu tel que le nombre de carte jaune, rouge, etc. Le jeu s'arrête quand on arrive au tour du dernier joueur qui a déjà pioché la dernière carte du jeu ou quand le nombre de jetons rouge est égale à 3 ou aussi quand la méthode qui calcule le score retourne 25 en affichant "partie terminée".

La classe Player : Elle interagit avec la classe *Card* pour piocher des cartes dans la main d'un joueur. Il s'agit d'une classe abstraite où il y a deux classes qui héritent de cette classe. Les deux classes qui héritent de la classe *Player* sont : *CliPlayer* représentant un joueur ordinaire et la classe *FirstCardBot* qui représente un robot et qui hérite de la classe abstraite *Bot* qui elle-même hérite de la classe *Player*. Selon le type de joueur, son rôle est aussi de choisir une action pour jouer au jeu tel que jouer une carte où le joueur va voir ensuite s'il a joué la bonne carte ou s'il doit défausser la carte . Il y a aussi l'action de défausser une carte où il récupère de la classe *Game* la défausse pour stocker une autre carte et incrémente le nombre de jetons bleus puis ajoute une nouvelle carte dans sa main . Le joueur peut aussi donner

une information et il va accéder au paramètre de la classe *Game* pour voir par rapport au nombre de jetons bleus s'il peut donner une information et s'il peut le faire, il doit choisir entre donner une information sur la couleur ou sur la valeur d'une carte. Quand il s'agit d'un robot qui joue, l'exécution du jeu se lance sans interaction avec l'utilisateur joueur contrairement au joueur ordinaire.

1. **La classe Card :** Elle représente une carte dans le jeu qui est identifiée grâce à la couleur et la valeur. Elle peut être masquée par le *currentPlayer* en affichant pour chaque tour du joueur ses cartes marquées par des X.

2. **L'énumération Color :** Elle permet d'identifier toutes les couleurs des cartes du jeu.

3. **La classe Cli :** Elle représente la classe où se trouve la méthode *main*. Elle instancie la classe *Game*.

Je vais maintenant expliquer ici le fonctionnement de l'utilisation de ce jeu en mode console.

Pour tester ce jeu en mode console, il faut l'exécuter et tester les actions possibles du joueur. Si l'on veut jouer une carte et qu'il s'agit d'un joueur ordinaire il faut taper *play* suivi de la valeur de la carte jouée. Si le joueur ordinaire veut défausser une carte, il faut taper *discard* suivi de la valeur de la carte que le joueur ordinaire veut défausser. Si l'on veut donner une information, il faut taper soit *hintv* suivie du numéro du joueur et suivie du numéro de la carte pour donner une information sur la valeur des cartes du joueur soit il faut taper *hintc* suivie du numéro du joueur suivie de la couleur pour donner une information sur la couleur des cartes du joueur.

Je vais enfin vous montrer ici des exemples pour l'utilisation de ce jeu en mode console :

1. **Play 0** –> le joueur courant est entrain de jouer la carte en position 0 de sa main.

2. **discard 1** –> le joueur courant est entrain de défausser la carte en position 1 de sa main.

3. **hintv 2 1** –> le joueur courant est entrain de donner une information sur la valeur 1 de carte au joueur en position 2 du jeu.

4. **hintc 2 BLUE** –> le joueur courant donne une information sur la couleur bleu des cartes que possède le joueur en position 2 du jeu.

Voici dans la figure 9 et 10, deux scénarios possible avec le jeu. Le premier illustre une version où il n'y aurait que quatre joueurs. Ce scénario représente la fin d'une partie d'Hanabi. Le second illustre qu'il n'y a que des joueurs robots qui jouent. Il est bien sûr envisageable de mixer les deux types de joueur. Pour l'instant, ces joueurs robots ne sont sont pas dotés d'intelligence mais il sera possible dans une prochaine version de ce jeu de les programmer pour utiliser une stratégie qui pourrait aider l'équipe à gagner plus facilement.

```
[2 WHITE, 1 BLUE, 1 WHITE, 3 WHITE]
[1 BLUE, 4 WHITE, 2 WHITE]
[2 BLUE, 3 BLUE, 1 WHITE, 3 WHITE]
[X XXX,X XXX,X XXX,X XXX]
hintv 2 1
nouveau tour de table
joueur suivant
blueTocken : 7 redToken : 1
pile : 0
defausse : 2
blanc : 0
jaune : 0
rouge : 0
bleu : 0
vert : 0
[X XXX,X XXX,X XXX,X XXX]
[1 BLUE, 4 WHITE, 2 WHITE]
[2 BLUE, 3 BLUE, 1 WHITE, 3 WHITE]
[5 WHITE, 1 WHITE, 1 BLUE, 2 BLUE]
play 1
La pile est vide
partie terminée 1
```

Figure 9: La version joueur

```
[3 WHITE, 3 WHITE, 1 WHITE, 4 WHITE]
[1 BLUE, 2 BLUE, 3 BLUE]
[X XXX,X XXX,X XXX,X XXX]
[1 BLUE, 1 WHITE, 1 WHITE, 2 WHITE]
La pile est vide
joueur suivant
blueTocken : 8 redToken : 2
pile : 0
defausse : 2
blanc : 0
jaune : 0
rouge : 0
bleu : 1
vert : 0
[3 WHITE, 3 WHITE, 1 WHITE, 4 WHITE]
[1 BLUE, 2 BLUE, 3 BLUE]
[3 BLUE, 4 WHITE, 2 BLUE]
[X XXX,X XXX,X XXX,X XXX]
La pile est vide
partie terminée 0
```

Figure 10: La version robot

5.2 Deuxième version en mode graphique

Pour développer une version graphique du jeu Hanabi, j'ai utilisé la bibliothèque libgdx[1]. C'est une biliothèque open-source multiplateforme destinée au développement de jeu vidéo. Elle est écrite en Java et permet aux développeurs de créer des jeux en utilisant le langage de programmation Java. Grâce à sa nature multiplateforme, elle permet également aux développeurs de créer un jeu unique et de le déployer sur plusieurs plateformes sans avoir à réécrire du code.

La première version de cette interface ne prenait en compte que la possibilité d'avoir que des joueurs de certitude comme Intelligence Artificielle et un joueur humain en option. La figure 11 montre l'écran de départ du jeu en mode graphique.

Dans cette interface, on voit tout d'abord que quand les cartes sont grises, cela représente un joueur humain qui ne doit pas voir ses cartes. Il y a ensuite les 2 boutons en dessous de chaque carte du joueur courant dont le 1er bouton sert à jouer une carte et le 2ème bouton permet de jeter une carte.

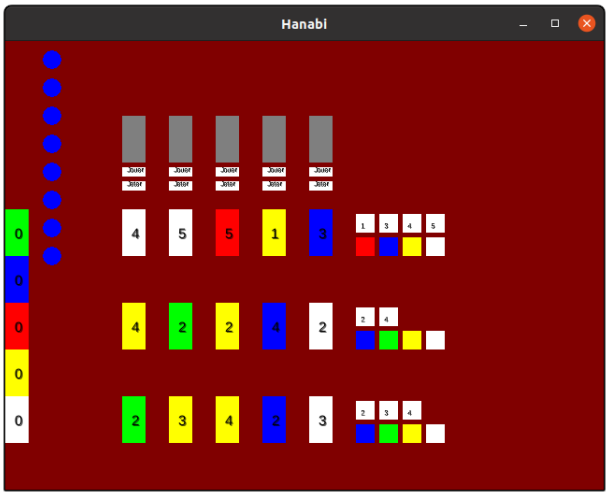

Figure 11: La version graphique du jeu Hanabi

Dans cette interface, on voit également trois autres joueurs qui représentent des joueurs d'IA de type joueur de certitude. Ils ont des boutons pour donner un indice à droite de chaque joueur. Les boutons colorés en blanc servent à donner un indice de valeur. Ici par exemple, le joueur courant peut donner un indice de valeur au joueur 1 sur la carte de valeur 1 ou 3 ou 4 ou

[1] https://libgdx.com/

5 ou bien il peut donner un indice de valeur sur l'un des autres joueurs en faisant le même fonctionnement.

Il y a aussi des boutons de plusieurs couleurs qui servent à donner un indice de couleur. Ici par exemple, le joueur courant peut donner un indice de couleur au joueur 1 sur la carte de couleur rouge ou bleu ou jaune ou blanc ou il peut donner un indice de couleur sur l'un des autres joueurs en faisant le même fonctionnement.

On voit à gauche le plateau avec les cinq couleurs représentant les différents niveaux de feu d'artifice. Ici, ils sont à zéro puisque c'est le début du jeu. Les jetons bleus au nombre de huit permettent de donner un indice.

Dans la figure 12, après avoir donné un indice, les joueurs de certitude ont donné également des indices. On voit que le joueur humain a trois cartes vertes dont l'une de valeur 1 complète le plateau. Il a également une carte rouge de valeur 1 qui complète le plateau. Sa dernière carte est un 1 mais sans l'indice de couleur. Le plateau est pour l'instant toujours à zéro car personne n'a joué une carte. De même les jetons bleus ne sont plus que 4 car les quatres joueurs ont donné un indice.

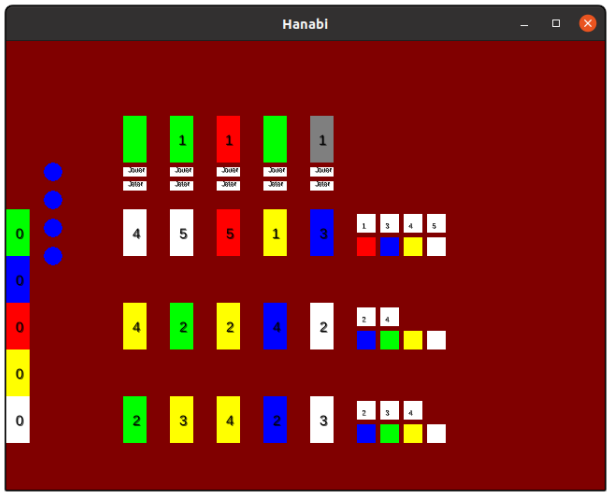

Figure 12: Premier coup

Dans la figure 13, on voit que le joueur humain a joué la carte rouge de valeur 1. Le plateau a donc pu être complété par la valeur 1 pour la couleur rouge. Les jetons bleus ont diminué car les autres joueurs ont donné un indice.

Dans la figure 14, après plusieurs coups, la partie est terminée et le jeu affiche le score final de 10.

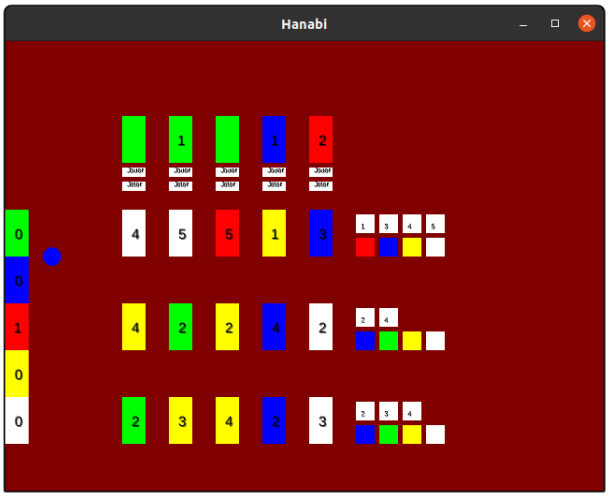

Figure 13: Deuxième coup

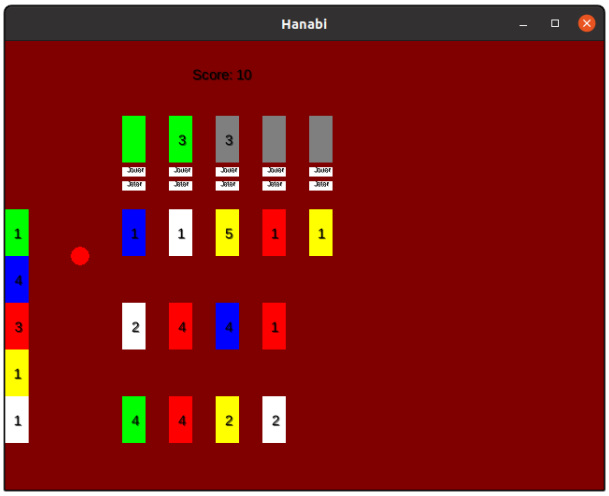

Figure 14: Fin de la partie

6 L'algorithme d'IA du Joueur de certitude

6.1 Fonctionnement du joueur de certitude

Le joueur de certitude est un joueur qui ne prend pas de risque. Il ne joue une carte uniquement que s'il a eu toutes les informations sur cette carte, c'est à dire sa couleur ou sa valeur. De la même manière,il ne défausse une carte que lorsqu'il est sûre que sa carte soit jetable ce qui veut dire qu'il a eu toutes les informations pour défausser la carte. Le joueur de certitude donne

en priorité des informations aux autres joueurs. Lorsqu'il manque des jetons bleus, ou qu'il n y a pas d'informations à donner, il joue ou défausse une carte qu'il est sûr de pouvoir jouer ou jeter. S'il n'a pas pu donner d'indices, ou jouer/jeter une carte avec certitude, alors il jette la plus vieille carte de sa main. Bien que le joueur soit présenté par Bouzy dans son article, il a omis de nombreux détails d'implémentation. En particulier il y a des ambiguités sur la certitude de certaines situations. Par exemple, si toutes les couleurs sont au niveau 1, et qu'un joueur a une carte de valeur 2 et de couleur inconnu, cette carte peut elle être jouée avec certitude ? De tels ambiguités, limite la reproducticibilité des résultats de Bouzy. L'implémentation des joueurs de certitude dans ce projet peut être différent de la sienne.

6.2 Implémentation du joueur de certitude

Pour l'implémentation, j'ai créé une classe **CertaintyPlayer** qui représente le joueur de certitude.

Dans cette classe, la méthode **shouldGiveInfo** vérifie si une carte est jouable ou jetable. Si c'est le cas, elle renvoie vrai pour que le joueur de certitude puisse en informer le joueur possesseur de cette carte.

La méthode **play** est la méthode principale de l'implémentation du joueur de certitude. Si le joueur de certitude a encore des jetons bleus, il va d'abord essayer de donner un indice en parcourant les cartes des autres joueurs et en appelant la méthode **shouldGiveinfo**. Il peut dans ce cas, donner un indice de type couleur ou de type valeur en utilisant les méthodes **giveHintColor** ou **giveHintValue**.

Au cas où le joueur de certitude n'a pas pu donner un indice, il va essayer soit de jouer l'une de ses cartes, soit d'en jeter une. Pour cela, il va appeler la méthode **canPlayCard** qui permet de vérifier qu'une carte est jouable. Si ce n'es pas le cas, il va appeler la méthode **canDiscardCard** qui permet de vérifier qu'une carte est jetable.

Si le joueur n'a pu ni donner d'information ni jouer sa carte ou la défausser, alors il va défausser la carte la plus ancienne de sa main. La figure suivante résume la démarche du joueur de certitude.

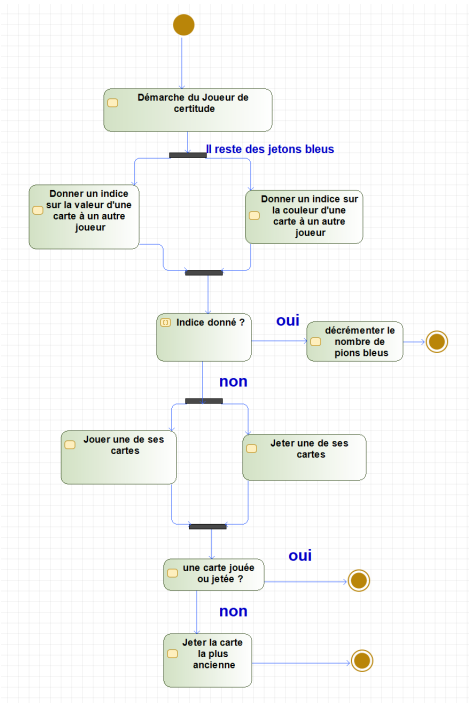

Figure 15: Démarche du joueur de certitude

7 L'algorithme d'IA de la stratégie de recommandation du chapeau

7.1 Fonctionnement de la stratégie de recommandation du chapeau

Le principe du chapeau permet d'obtenir de très bons scores, souvent proches de 25. Il y a deux stratégies possibles qui utilisent le principe du chapeau, la première est la stratégie de recommandation et la seconde est la stratégie d'information [COX2015]. J'ai choisi la stratégie de recommandation. L'idée sous-jacente de cette stratégie est de transmettre une recommandation à tous les autres joueurs, en donnant un seul indice. Il faut donc un algorithme pour encoder ces recommandations en un seul indice, puis un autre algorithme pour que chaque joueur puisse retrouver leur recommandation à partir de l'indice donné. Cette stratégie est implémentée en représentant la main d'un joueur par un "chapeau", c'est-à-dire un nombre h tel que $0 <= h < H$. Dans la stratégie de recommandation, $H = NCPP * 2$. NCPP représente le nombre de cartes par joueur. Le chapeau h d'un joueur "recommande" un coup au joueur lorsque $h < NCPP$, la recommandation est de "jouer la carte numéro

h" en commençant par la gauche. Dans le cas contraire, la recommandation est de "défausser la carte numéro h - NCPP" en commençant par la gauche. L'article de Bouzy mentionne un programme s'appelant RECOMPROG pour calculer un chapeau. Mais n'ayant pas réussi à le trouver, j'ai choisi de l'implémenter à partir des règles suivantes décrites dans l'article de Cox :

- Recommander de jouer une carte de valeur 5 de plus petit index dans la main si elle complète un feu d'artifice
- Recommander de jouer une carte de plus petite valeur et de plus petit index si elle complète un feu d'artifice
- Recommander de jeter une carte de plus petit index qui n'est pas jouable
- Recommander de jeter une carte de plus grande valeur qui n'est pas indispensable
- Recommander de jeter la carte d'index 1

Un joueur spécifique voit les mains des autres joueurs. Par conséquent, il peut calculer leurs chapeaux. Communiquer avec le principe du chapeau consiste à utiliser les coups d'information d'Hanabi pour transmettre la somme des chapeaux que le joueur voit. Lorsqu'un joueur observe un coup d'information effectué par un joueur donné, il peut calculer la valeur de son propre chapeau par différence entre la somme des chapeaux transmis dans le cadre du coup d'information et la somme des chapeaux qu'il voit (sauf le chapeau du joueur donné).

Le joueur actif doit transmettre la somme des chapeaux aux autres joueurs. Il doit encoder cette somme en un indice. Dans l'article de Cox, l'encodage retourne un joueur destinataire de l'indice et un type d'indice (soit n'importe quelle couleur soit n'importe quelle valeur). Dans l'article de Bouzy, l'encodage retourne un joueur destinataire de l'indice, et un indice spécifique(soit une couleur spécifique, soit une valeur spécifique). J'ai décidé d'implémenter la version de Cox. Pour encoder et décoder la somme des chapeaux, deux fonctions sont nécessaires. Avec un code S tel que $0 <= S < H$, **Code2Couple** produit un couple (B, I) où I est l'information à envoyer au joueur B par le joueur A. Avec un couple (B, I), **Couple2Code** produit un code S. Lorsque le joueur A veut donner une information, il calcule S, la somme des chapeaux qu'il voit modulo H, et il informe avec **Code2Couple(S)=(B ; I)**. Par conséquent, les autres joueurs voient **(B ; I)** et en déduisent **Couple2Code(B ; I) = S**, la somme des chapeaux vus par A. Donc, chaque joueur, différent

de A, voyant tous les chapeaux vus par A sauf le sien, peut calculer la valeur de son propre chapeau.

Quand un joueur recevra une recommandation, il devra prendre en compte un ordre de priorité d'actions qui sont les suivantes :

1. Si la recommandation la plus récente était de jouer une carte et qu'aucune carte n'a été jouée depuis le dernier indice, jouez la carte recommandée.

2. Si la recommandation la plus récente était de jouer une carte, une carte a été jouée puisque l'indice a été donné, et que les joueurs ont fait moins de deux erreurs, jouez le Carte recommandée.

3. Si les joueurs ont un jeton indice, donnez un indice.

4. Si la recommandation la plus récente était de jeter une carte, jetez la carte demandée.

5. Jetez la carte c1.

Le principe du chapeau est puissant en ce sens qu'un coup d'information informe tous les joueurs en même temps, et pas seulement le joueur ciblé. Par conséquent, les jetons bleus peuvent être plus fréquemment économisés.

7.2 Implémentation de la stratégie de recommandation du chapeau

Pour l'implémentation, j'ai crée une classe **HatRecommendation** qui représente le principe du chapeau basée sur la stratégie de recommandation.

Dans cette classe la méthode **play** décide de l'action du joueur. Elle va tout d'abord vérifier si le joueur connait son chapeau, et si c'est une recommandation de jouer une carte. Dans ce cas le joueur va essayer de jouer.

Sinon si cette première condition n'est pas valide, la méthode va tenter de communiquer les chapeaux s'il reste des jetons bleus, en appellant la méthode **calculHat**. Le joueur courant parcourt toutes les cartes que possède chaque joueur et applique les règles de priorité suivantes :

- Le joueur courant recommande à l'autre joueur de jouer une carte de valeur 5 de plus petit index de la main si elle complète le plateau.

- Si la 1ère règle ne marche pas, le joueur courant recommande à l'autre joueur de jouer une carte de plus petite valeur de plus petit index de la main si elle complète le plateau.

- Si la 1ère et la 2ème règle ne marchent pas, le joueur courant recommande à l'autre joueur de jeter une carte de plus petit index de la main qui n'est pas jouable.
- Si la 1ère, la 2ème et la 3ème règle ne marchent pas, le joueur courant recommande à l'autre joueur de jeter une carte qui n'est pas indispensable (autre carte que valeur 5).
- Si aucune de ces règles n'a marché, il recommande de jeter la carte la plus ancienne.

Une fois la valeur du chapeau calculée, il va falloir coder grâce à la méthode **code2Couple** (voir figure 16) la somme des chapeau des joueurs en lui appliquant le modulo du nombre de cartes par joueur * 2, qui va retourner un couple (type d'information (valeur ou couleur) + position du joueur à laquelle il faut donner un indice). Si cette valeur est compris entre 0 et la taille de la main du joueur visé - 1, alors il s'agit d'une recommandation de jouer une carte. Sinon si cette valeur est supérieure à la taille de la main du joueur visé - 1, alors il s'agit d'une recommandation de jeter une carte. Selon le type d'information, il faudra ensuite appeler la méthode **giveHintColor** si c'est une couleur, ou **giveHintValue** si c'est une valeur.

```java
public HatCouple code2Couple(int sumHat){
int ncpp = getGame().getNcpp();
int np = getGame().getPlayers().size()-1;
int data = sumHat % (ncpp * 2);

    HintType type = data < np?HintType.HINT_VALUE:HintType.HINT_COLOR;
    Player dest = getGame().getNextPlayers().get(data < np?data:data-np>;
    return new HatCouple(dest, type);
}

public int couple2Code(HatRecomendation.HatCouple couple){
    int indiceJoueur = getGame().getNextPlayers().indexOf(couple.dest);
    if(couple.type == HatRecomendation.HintType.HINT_COLOR){
        indiceJoueur += getGame().getNextPlayers().size();
    }
    return indiceJoueur;
}
```

Figure 16: Implémentation de la méthode code2Couple et couple2Code en Java

Ensuite, la méthode **notify** (voir fig 17) sera appelée sur chaque autre joueur pour qu'il puisse calculer leur propre chapeau. Dans cette méthode

notify, les joueurs décodent la valeur du chapeau global estimé du joueur qui a fait la recommandation en appelant la méthode **Couple2Code** (voir figure 16).

```
public void notify(Player source, Player dest, Color c, Integer v){
 int otherPlayerHat = 0;
 int sumHat = couple2Code(new HatRecomendation.HatCouple(dest, v == null ?
    HatRecomendation.HintType.HINT_COLOR: HatRecomendation.HintType.HINT_VALUE));
 for(Player p : getGame().getPlayers()){
    if(p!=source && p!=this){
        otherPlayerHat += calculHat(p);
    }
 }
 lastHat = sumHat - otherPlayerHat;
 while(lastHat < 0) {
    lastHat += getGame().getNcpp() * 2;
 }
    System.out.println("--- player " + this.getId() + " computed hat :"+ " " + lastHat);
}
```

Figure 17: Implémentation de la méthode notify en Java

Dans cette méthode, il s'agit de calculer la valeur correspondant au couple (type d'information(valeur ou couleur) + position du joueur à laquelle il faut donner un indice). Si le type d'information correspond à une valeur, alors elle récupère à partir de la position prochaine du joueur courant le joueur dont la position est indicée et retourne cette valeur là. Sinon si le type d'information correspond à une couleur, alors il additionne l'indice du joueur qui a donné la recommandation avec le nombre de joueur qu'il y a autour et retourne ce résultat là. Il s'agit de retourner le résultat du décodage de l'information qui a été donné, c'est à dire le chapeau total.

Une fois que la méthode **notify** a calculé la somme des chapeaux des autres joueurs, elle calcule ensuite son propre chapeau en faisant la soustraction de la valeur du chapeau total estimé du joueur qui a fait la recommandation, moins la somme des chapeaux des autres joueurs. Si la valeur trouvée est négative, alors ils rajoutent à leur valeur du dernier chapeau trouvé(**lastHat**) le nombre de cartes par joueur * 2 jusqu'à ce que la valeur du dernier chapeau(**lastHat**) du joueur soit supérieur ou égal à 0.

Finalement, s'il n'est pas possible de jouer une carte ou de faire une recommandation, alors le joueur va essayer de jeter une carte. Si son chapeau lui recommande de jeter une carte, il la jette. Sinon il jette la plus ancienne carte de la main.

Pour permettre de tester un joueur basé sur le principe de la stratégie de recommandation du chapeau, l'interface graphique a été améliorée en

affichant la valeur du chapeau de chaque joueur comme le montre la figure 18.

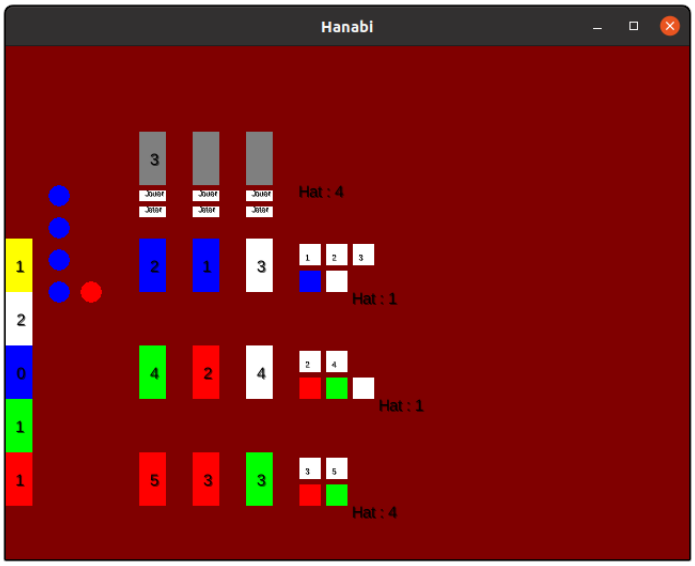

Figure 18: Interface graphique du joueur de chapeau

8 Nos résultats

J'ai créé des programmes qui permettent de lancer une simulation sur une centaine de parties pour chaque algorithme d'IA, et chaque combinaison de nombre de joueurs et nombre de cartes. La première colonne représente le nombre de joueurs. La seconde colonne est le nombre de carte par joueur et enfin la troisième colonne représente la moyenne des scores sur cent parties. Le fait que le score soit à null est l'indicateur que cette configuration n'est pas gérable dans notre application. J'ai constaté qu'en augmentant le nombre de parties, les résultats n'évoluaient pas énormément.

Dans la simulation avec des joueurs utilisant la stratégie de recommandation, certaines configurations fonctionnent mieux que d'autres comme la configuration 4 joueurs et 3 cartes par joueur. Les autres configurations sont moins probantes. Les valeurs sont stables si on relance plusieurs fois de suite la simulation. La stratégie de recommandation ne fonctionne que pour certaine combinaison de joueur / carte par joueur. Pour X joueurs, le joueur actif a x-1 destinataires d'information, donc code2Couple peut retourner 2 * (x-1) couples différents. Comme ce couple peut encoder une recommandation de jeu ou de défausse, il peut y avoir au maximum X-1 cartes par joueur. La

version de Bouzy n'a pas ce problème car son code2Couple peut retourner son couple. Cela lui permet d'avoir de meilleurs résultats.

player type : HAT RECOMENDATION

4,3:16.68

5,3:11.81

5,4:10.7

Les valeurs concernant le joueur de certitude sont dans l'ensemble correctes. Elles sont néanmoins différentes que celles présentées dans l'article de Bouzy [BOU 2017]. En particulier, nos scores faibles sont plus élevés que les siens (par exemple, avec 2 joueurs et 3 cartes, nous obtenons 12.43 au lieu de 10.31). Au contraire, nos scores élevés sont plus faibles que les siens (avec 5 joueurs et 3 cartes, nous obtenons 10.37 au lieu de 15.21). Cet écart peut s'expliquer par des différences d'implémentation, l'article de Bouzy omettant tout détail algorithmique.

player type : CERTAINTY

2,3:12.43

2,4:12.98

2,5:13.62

3,3:11.5

3,4:11.95

3,5:12.06

4,3:11.08

4,4:10.92

4,5:11.02

5,3:10.37

5,4:10.17

5,5:9.6

9 Conclusions

Ce projet a été très enrichissant car il m'a permis d'approfondir mes connaissances en Intelligence Artificielle. Mon étude bibliographique et l'analyse que j'ai faite sur ce projet m'ont aidé à avoir des perspectives sur les approches possibles pour développer des joueurs d'IA. D'ailleurs, la lecture d'articles de recherche a été pour moi une initiation très interessante pour envisager une implémentation d'algorithmes d'IA dans le jeu de Hanabi. J'ai appris à confronter ces articles, et à exploiter leur complémentarité. Les deux articles qui m'ont permis de réaliser ce travail ont été très utiles et très différents dans leur apport. Le premier [BOU2017] fait une très bonne synthèse du principe

du chapeau, mais ne détaille pas l'implémentation de la stratégie de recommandation que décrit très bien le deuxième article [COX2015]. L'approche mathématique du principe du chapeau utilisée pour aborder un problème d'Intelligence Artificielle a été vraiment très enrichissante. Maintenant, son implémentation a été difficile mais m'a permis de mieux comprendre les articles de recherche que j'avais lu dans la première période de mon travail.

Le jeu Hanabi est vraiment un excellent terrain d'études de part l'idée qu'un ensemble de joueurs unissent leur analyse du jeu pour gagner ensemble. D'autres pistes d'améliorations pourraient être envisagées comme l'algorithme de Monte-Carlo qui est une méthode statistique lorsque les approches déterministes traditionnelles sont difficiles à appliquer comme pour le jeu Hanabi. Cette approche pourrait compléter les algorithmes utilisant le principes du chapeau en exploitant une combinaison de recherche en profondeur et d'échantillonnage aléatoire pour explorer l'arbre du jeu.

References

[BAR2019] Nolan Bard et al, *The Hanabi Challenge: A New Frontier for AI Research, 2010 MSC: 68T01, 2019*

[BAU2017] Antoine Bauza, *Présentation des règles du jeu, https://www.youtube.com/watch?v=pr8gvJBUgQk, 2017*

[BAU2021] Antoine Bauza et al , *Table ronde Hanabi, défier l'IA, https://www.youtube.com/watch?v=UEENb-YJWBI, 2021*

[BOU2017] Bruno Bouzy , *Playing Hanabi Near-Optimally, Université Paris Descartes , 2017*

[COX2015] Cox, *How to Make the Perfect Fireworks Display Two Stratégies for Hanabi, 2015*

[CRA2017] Crazysmiler, *Méthode Hanabi, https://www.trictrac.net/forum/sujet/hanabi-methode, 2017*

[DEE2021] Deepmind, *Hanabi learning environment, https://github.com/deepmind/hanabi-learning-environment, 2021*

[GIO1991] Giove,*Une implémentation d'Hanabi en Python, https://github.com/giove91/hanabi, 1991*

[GRO2021] Grooten et al , *Deep Reinforcement Learning for the cooperative card game Hanabi, Eindhoven University of Technology, 2021*

[LEG2018] Bénédicte Legastelois , *Extension pondérée des logiques modales dans le cadre des croyances graduelles, HAL open science, 2018*

[LEM2019] Le monde, *Intelligence artificielle : DeepMind s'intéresse au jeu de cartes français Hanabi, https://www.lemonde.fr/pixels/article/2019/02/06/intelligence-artificielledeepmindsinteresseaujeudecartesfrancais-hanabi_5420186_4408996.html, 2019*

[RIV2017] Joseph Walton-Rivers et al, *Evaluating and modelling Hanabi-playing agents, 2017*

[SAU2022] Guilhem Saurel, *Une implémentation d'Hanabi en Rust, https://gepgitlab.laas.fr/recreation/hanabi, 2022*

[TEC2021] Technologie and Science, *Une étude du MIT révèle que les humains luttent lorsqu'ils sont en partenariat avec des agents RL, https://lesactualites.news/technologie-et-science/une-etude-du-mit-revele-que-les-humains-luttent-lorsquils-sont-en-partenariat-avec-des-agents-rl/, 2021*

[RIV2017] Joseph Walton-Rivers et al, *Evaluating and modelling Hanabi-playing agents, 2017*

[TRU2022] Bryan Truong, *Critique du jeu de cartes Hanabi, https://gamecows.com/fr/critique-du-jeu-de-cartes-hanabi/, 2022*

[VIN2022] vincent et mister Lou, *Une partie de Hanabi (2 joueurs), https://gamecows.com/fr/critique-du-jeu-de-cartes-hanabi/, 2022*