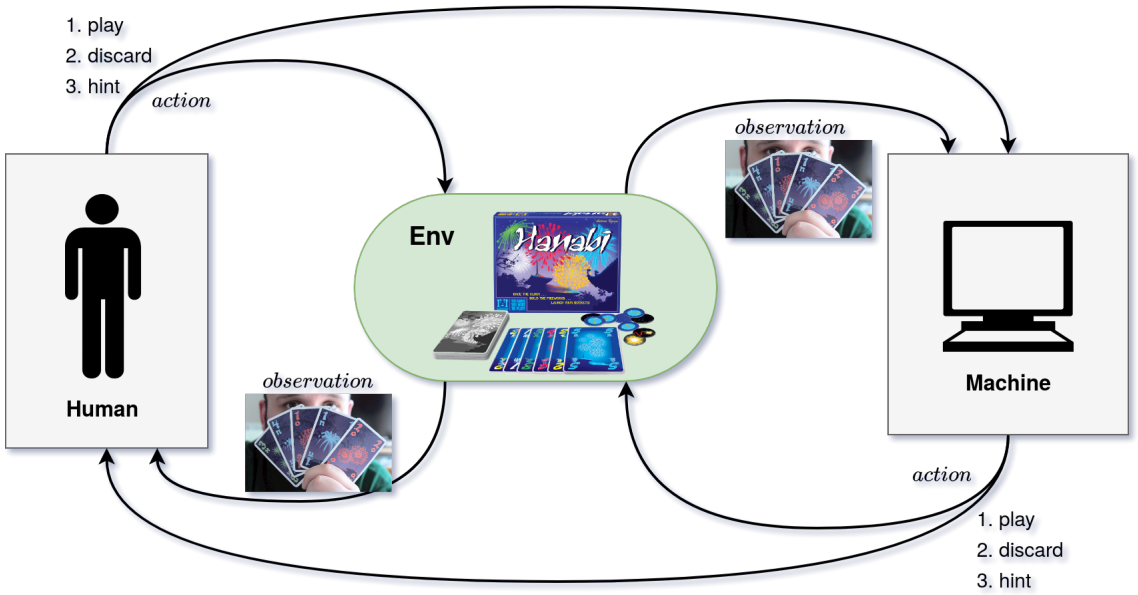
INSA ROUEN NORMANDIE

RAPPORT DE STAGE INGÉNIEUR

Collaboration Humain-Machine



DAVID ALBERT
GM5 - PROMO 2020

A l'attention de :

M. JILLES STEEVE DIBANGOYE
M. LAURENT VERCOUTER
MME NATHALIE CHAIGNAUD
MME CÉCILIA ZANNI-MERK

23 mars 2020 – 28 août 2020

Remerciements

Je voudrais tout d'abord remercier Jilles Steeve Dibangoye pour le temps qu'il a consacré à m'expliquer les différents aspects théoriques du sujet. Merci à lui et à Laurent Vercouter de m'avoir guidé tout au long de ce stage sur les différentes tâches ainsi que de m'avoir fait découvrir de nombreuses choses concernant leur domaine d'expertise. Le temps passé à comprendre les résultats d'apprentissage et discuter des modèles implémentés m'a fortement aidé à cerner le sujet et ses difficultés.

Je remercie aussi ma tutrice Nathalie Chaignaud pour son suivi et son écoute en cas de problème.

Table des matières

Introduction

1 Contexte du stage

Le stage de fin d'étude est l'étape finale avant l'obtention du diplôme d'ingénieur. En tant qu'étudiant en stage, l'objectif est multiple. Il vise à la fois à se familiariser avec un environnement de travail professionnel, à découvrir un poste qui pourrait être occuper par la suite, à monter en compétence et à entamer l'insertion professionnelle.

Dans cette optique, j'ai décidé de réaliser ce stage au sein de deux laboratoires de recherche universitaire. Le premier est le LITIS [1]. Il est situé à Rouen et est sous la tutelle de l'INSA de Rouen. Le second, quant à lui, est un laboratoire de l'INSA de Lyon qui se nomme le CITI [2]. Le stage a été financé par l'INSA dans le cadre d'un projet visant à promouvoir la recherche inter-INSA. Bien qu'intégré à deux équipes, la localisation principale a été le site de Lyon. Dans le contexte actuel de crise sanitaire [3], le stage a entièrement été réalisé en télétravail.

1.1 Contexte au LITIS

Le Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes, ou LITIS, est un laboratoire académique associé à l'INSA de Rouen. Au sein du LITIS, j'ai fait parti de l'équipe MIND, qui s'intéresse à l'interaction d'agents virtuels avec un utilisateur humain sous différents aspects (agents conversationnels, système de recherche documentaire personnalisé, etc). Cette équipe est donc amenée à développer des modèles d'interaction et de décision pour subvenir à ses besoins.

1.2 Contexte au CITI

D'autre part, le Centre d'Innovations en Télécommunications et Intégration de Services, ou CITI, est un laboratoire académique associé à l'INSA de Lyon et à l'INRIA. J'ai effectué mon stage au sein de l'équipe CHROMA dont les principales thématiques de recherche sont la navigation et la planification en robotique.

2 Objectif du stage

Etant rattaché à deux équipes, les recherches réalisées durant le stage doivent avoir un intérêt pour les deux parties. De cette façon, le sujet s'est construit autour des thématiques de planification et de décision dans le cadre d'un agent logiciel collaborant avec un utilisateur humain. L'agent logiciel et l'utilisateur évoluent dans un même environnement et doivent accomplir une tâche bénéfique à l'humain. Durant ce stage, nous chercherons à élaborer un algorithme permettant de contrôler les actions de la machine en vue de cette collaboration. Nous nous focaliserons sur une approche par

1. https://www.litislab.fr
2. http://www.citi-lab.fr
3. Période de crise sanitaire liée au COVID-19.

apprentissage par renforcement pour construire la politique d'action de l'agent logiciel et évaluerons notre algorithme sur la plateforme Hanabi (voir 1.2).

La cadre de la collaboration humain-machine est un cadre complexe qui soulève de nombreuses questions et difficultés. Par exemple, le terme "humain", comme il est employé ici, ne fait pas référence à une personne unique mais à l'ensemble des humains. Notre agent logiciel devra être capable de collaborer avec des utilisateurs très différents, qui n'ont pas tous la même façon de penser et d'agir. D'autre part, nous souhaitons que ce soit la machine qui s'adapte à l'humain et non pas l'inverse comme c'est généralement le cas dans les applications actuelles. L'humain n'aurait plus besoin de comprendre le fonctionnement de la machine et de s'adapter à celle-ci et pourrait garder son comportement normal (comportement qu'il aurait eu avec d'autres humains).

Il est à de noter que ce type de collaboration entre l'humain et la machine a de nombreux domaines d'application. Parmis les applications intéressantes pour le LITIS, nous pouvons citer la réalisation d'un agent conversationnel collaboratif utilisé pour une tâche de recherche d'information ou alors pour un système de questions-réponses. Quant au CITI, un tel algorithme pourrait être utilisé lors d'une tâche de navigation collaborative en robotique ou avec des véhicules autonomes.

Prenons ce dernier exemple pour illustrer le problème général. Dans quelques années, les humains devront partager les routes avec des voitures dont la conduite est autonome. Dans de nombreuses situations, telles que celle présentée sur la figure 1, la machine aura la nécessité de prendre des décisions qui impliqueront un ou plusieurs humains. Dans ce genre de situation, nous voudrons que la machine ait un comportement collaboratif avec la personne en face. Toutefois, nous ne pouvons prédire de façon formelle le comportement du conducteur humain. Notre algorithme devra être en capacité de comprendre les intentions du conducteur humain grâce à des observations locales et de réagir en conséquence. Cette collaboration servira, par exemple, à prévenir d'éventuels dangers.

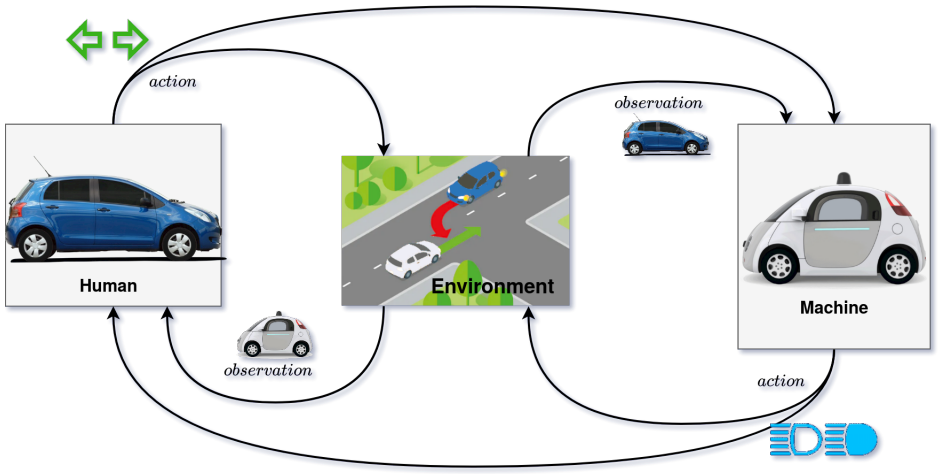


FIGURE 1 – Dynamique du système étudié dans le cadre de la voiture autonome.

Partie I

Cadre général

1 Introduction aux problématiques

Dans l'introduction nous avons brièvement présenté l'objectif du stage. Nous allons maintenant décrire plus précisément les différents aspects du sujet.

1.1 Collaboration humain-machine

On rencontre plusieurs problématiques quand on aborde le sujet de la collaboration entre humains et machines. Deux problématiques particulièrement importantes dans notre contexte sont la communication et le facteur humain. Il faut en effet définir ces termes dans un cadre précis.

1.1.1 La communication

En général, la résolution d'une tâche collaborative nécessite un moyen de partage d'informations entre les agents. Pour être comprise par tous les agents, l'information communiquée a besoin de suivre des règles connues par tous. Les humains ont, au fil du temps, su élaborer un ensemble de conventions formant un langage mais sa grande complexité reste difficile à utiliser pour les machines et est un problème à part entière. Nous ne traiterons pas ce problème dans ce rapport et nous considérerons que les agents partagent un langage commun. D'autre part, l'information en question a besoin d'un canal de transmission pour atteindre les différentes entités. Ainsi, dans beaucoup d'applications, communiquer a un coût (le coût de transport de l'information via le canal) et ne peut pas être utilisé sans limites. La plateforme Hanabi, qui sera présentée par la suite, a cette propriété.

1.1.2 Le facteur humain

Notre objectif durant ce stage n'est pas de collaborer avec un agent virtuel mais avec un humain quelconque. Or, l'humain utilise un processus cognitif très complexe qui est propre à chacun. En d'autres termes, il y a autant de façons d'agir dans une situation que de manière de penser. La simulation d'un utilisateur humain est de ce fait un problème très difficile auquel nous devrons proposer une solution.

1.2 La collaboration dans Hanabi

"The Hanabi Challenge : A New Frontier for AI Research" de Bard et al. [1] est le papier qui a introduit le jeu Hanabi comme un challenge pour la communauté IA. Ce papier introduit deux challenges, qui sont les challenges *self-play learning* et *ad-hoc teams* (voir ci-après), et des résultats de références pour les futures recherches (annexe A.1). Il fournit également une implémentation du jeu Hanabi [2] dotée

d'une interface Gym [1] pour faciliter sa prise en main. Comme expliqué auparavant, nous avons choisi cette plateforme pour vérifier le bon fonctionnement des algorithmes implémentés. Elle fournit une observation vectorisée pour chaque agent, de dimension 783.

Les 783 valeurs comprennent notamment :
- Les mains des coéquipiers
- Des informations sur le plateau de jeu
- Des informations sur la défausse
- Des informations sur la dernière action
- Les connaissances communes

1.2.1 Le jeu Hanabi

L'intérêt d'étudier ce jeu est qu'il comporte des propriétés intéressantes et, qu'à ce jour, aucune solution connue n'est capable de jouer de façon optimale dans toutes les situations. Le jeu a été créé en 2010 par Antoine Bauza et a reçu le *Spiel des Jahres* [2] en 2013.

FIGURE I.1 – Jeu Hanabi

Les principales caractéristiques d'Hanabi sont les suivantes :
- **Collaboratif** : C'est la caractéristique principale de notre problème. Nous étudions les problèmes coopératifs. Hanabi en est un bon exemple. En effet, les joueurs doivent contribuer à la réalisation de piles de cartes ordonnées.
- **Information incomplète** : Dans le jeu Hanabi, nous possédons des cartes que nous ne connaissons pas initialement. Ce sont nos coéquipiers qui les voient et qui devront nous les faire connaître. De plus, une pioche de cartes mélangées est mise en place. Cela fait du jeu, un jeu à information incomplète.
- **Communication limitée** : La communication est primordial dans le jeu. Communiquer correspond à l'une des trois actions possibles mais cette communication a un coût. En effet, chaque acte de communication nécessite de retirer un des 8 jetons bleu.
- **Séquentiel** : Le jeu Hanabi a la propriété d'être séquentiel. C'est-à-dire que les joueurs jouent à tour de rôle.

1.2.2 Résolution d'Hanabi

Depuis plusieurs années, des équipes de recherche s'intéressent à la résolution du jeu Hanabi. Certains algorithmes développés ont déjà montré leur capacité à réaliser de très bons scores. Cependant, la majorité de ces méthodes ne s'intéressent qu'au challenge ***self-play***. C'est-à-dire au cas spécifique

1. Type d'interface utilisé pour interagir de façon normalisée avec des environnements de simulation [3]
2. Récompense la plus prestigieuse pour les jeux de sociétés - https://www.spiel-des-jahres.de/en/

où tous les joueurs partagent une politique d'action commune. Dans ce cas, la politique en question n'est, en général, pas robuste aux changements de coéquipiers et ne permet donc pas leur utilisation pour notre problème. Notre situation s'apparenterait plutôt au challenge ***ad-hoc***, qui vise à jouer avec n'importe quel coéquipier. [3].

Pour rappel, nous cherchons à construire une politique d'action pour jouer avec des humains. Ainsi, nous souhaiterions qu'elle soit robuste avec un grand nombre de coéquipiers.

2 Le cadre mathématique

Pour résoudre le problème, nous avons choisi d'étudier les méthodes d'apprentissage par renforcement. Ces méthodes s'appuient sur un formalisme mathématique bien précis que nous développons dans cette section. Dans notre cadre d'étude, nous utilisons habituellement une généralisation des processus de décision de Markov au cas d'environnements partiellement observables et à n agents.

2.1 (PO)MDP

Un processus de décision de Markov (MDP) est un processus stochastique qui permet de formaliser la notion d'agent interagissant avec un environnement. A chaque instant, l'agent observe l'état de l'environnement puis exécute une action. Cette action change l'état de l'environnement de façon stochastique et un signal de récompense est transmis à l'agent.

Formellement, un MDP est décrit par un tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$ avec :
- $\mathcal{S}$: espace d'états
- $\mathcal{A}$: espace d'actions
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$: fonction de transition probabiliste, i.e. $\mathbb{P}(S_{t+1} = s' \,|S_t = s, A_t = a)$
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \to [0, 1]$: récompense probabiliste

A tout instant t, les joueurs d'Hanabi ne connaissent pas de manière exacte l'état de l'environnement. Nous avons donc besoin d'une généralisation du MDP dans le cas d'un environnement partiellement observable. C'est le cas du processus de décision de Markov partiellement observable (POMDP). Un POMDP ajoute au MDP standard l'idée que l'agent n'a accès qu'à une observation et non pas à l'état complet du système. Pour ce formalisme, on ajoute donc un espace d'observation $\mathcal{O}$ et une fonction d'observation Ω au tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$.

2.2 Dec-POMDP

Pour bien définir notre problème, le processus stochastique doit faire intervenir non pas un mais n agents et contraindre le fait que la tâche soit collaborative. Cela nous amène à une seconde généralisation : les POMDP décentralisés ou **Dec-POMDP** (voir définition en annexe A.1).

Nous nous intéressons au problème de collaboration avec un seul utilisateur à la fois. Nous ne considèrerons donc que le cas de Dec-POMDP à deux agents.

3 Quelques approches scientifiques intéressantes

Une partie état de l'art a été réalisée pour avoir une vision à la fois plus large et plus précise des approches existantes au sein de différentes communautés scientifiques. Approches qui aideraient à la résolution de notre problème.

3. Dans [1], il est même dit à propos du challenge *ad-hoc* : "The ultimate goal is agents that are capable of playing with other agents or even human players."

3.1 Les recherches en apprentissage par renforcement

L'apprentissage par renforcement est un domaine d'étude qui connaît une forte hausse de popularité de nos jours. Son côté interactif est intéressant car il permet à l'agent d'apprendre des conséquences de ses actions sans connaissances à priori sur l'environnement. Dans notre cas, ce sont les approches de résolution de POMDP et les approches d'apprentissage multi-agents qui sont pertinentes à étudier.

3.1.1 Apprentissage par renforcement mono-agent

Les algorithmes d'apprentissage par renforcement étudiés et implémentés sont des variantes du Q-learning.

Q-learning Le Q-learning est un algorithme itératif qui utilise l'expérience de l'agent pour construire la politique d'action optimale dans le cadre d'un MDP. Contrairement à certaines méthodes qui cherchent à optimiser directement la politique d'action, le Q-learning construit une fonction appelée *fonction de valeur d'action optimale* et dont l'expression est donnée dans l'équation I.1. Pour se faire, une méthode de mise à jour itérative de l'estimation de cette fonction est utilisée. Plus de détails sont disponible en annexe A.3 et dans [4].

$$q_*(s,a) = \max_\pi \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \tag{I.1}$$

Cette fonction correspond à l'espérance du gain G_t étant donné un état s et une action a. Elle peut être utilisée pour déterminer l'action optimale. Le gain G_t est défini dans l'équation I.2. G_t peut donc être vu comme la récompense cumulée pondérée par le coefficient γ.

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{I.2}$$

Pour choisir le paramètre γ, nous devons nous poser la question de l'importance que nous voulons accorder aux récompenses à long terme. Cette idée vient du fait que l'humain préfère en général une récompense à court terme (perçue comme peu risquée) à une récompense lointaine (beaucoup plus risquée).

Deep Q-learning En pratique, l'algorithme classique du Q-learning est uniquement applicable dans le cas où l'espace d'états est fini et petit. Or, dans le cas de la plateforme Hanabi, l'espace d'états est trop grand. Un bon moyen de faire face à cette limitation est d'utiliser des réseaux de neurones artificiels pour approximer la fonction q_*. Le Deep Q-learning est la variante du Q-learning qui fait usage d'un approximateur de fonction par réseau de neurones pour évaluer q_*. Un algorithme précurseur est DQN [5].

Deep Recurrent Q-learning Jusqu'ici, nous avons considéré le cadre d'un MDP et donc nous supposions la possibilité d'avoir accès à l'état de l'environnement à tout instant. Cependant, dans notre cas d'étude, cette supposition est fausse car nous n'avons accès qu'à une observation de l'environnement. Dans ces circonstances, on souhaite garder en mémoire les observations passées pour mieux comprendre l'état de l'environnement. Pour se faire, on utilise un réseau de neurones dit *récurrent*. Celui-ci permet de prendre en compte toutes les observations passées et plus uniquement la plus récente. Ainsi, le Deep Recurrent Q-learning est la modification du Deep Q-learning pour la résolution d'un problème de décision de Markov partiellement observable.

DRQN [6] et R2D2 [7] sont deux principales adaptations de DQN pour les POMDP. Ces algorithmes

ont été étudiés et utilisés pour les premières expérimentations. Un schéma expliquant le fonctionnement de l'algorithme R2D2 est présenté en annexe A.4.

Des explications détaillées sur les méthodes d'approximation de la politique d'actions et les algorithmes Q-learning et DQN sont disponibles en annexe A.3 et dans [4].

3.1.2 Apprentissage par renforcement multi-agents

Bien que peu utilisé lors des expérimentations, le cas multi-agents est le plus général et donc a été une approche envisagée. Une classification non exhaustive des méthodes d'apprentissage multi-agents a été réalisée en annexe A.5 pour bien comprendre les problématiques liées à ce cadre d'étude et les solutions existantes.

L'apprentissage par renforcement multi-agents est utilisé dans le cas où nous souhaitons contrôler la politique d'action de tous les joueurs. Or, dans notre cas, nous ne cherchons qu'à contrôler les actions de la machine et non celles de l'humain. Nous pouvons relaxer le problème et considérer le coéquipier (l'humain dans notre cas) comme faisant partie intégralement de l'environnement. Ainsi, pour un coéquipier fixé, le problème revient à résoudre un POMDP. Le passage Dec-POMDP à POMDP classique est schématisé dans la figure I.2 ci-dessous.

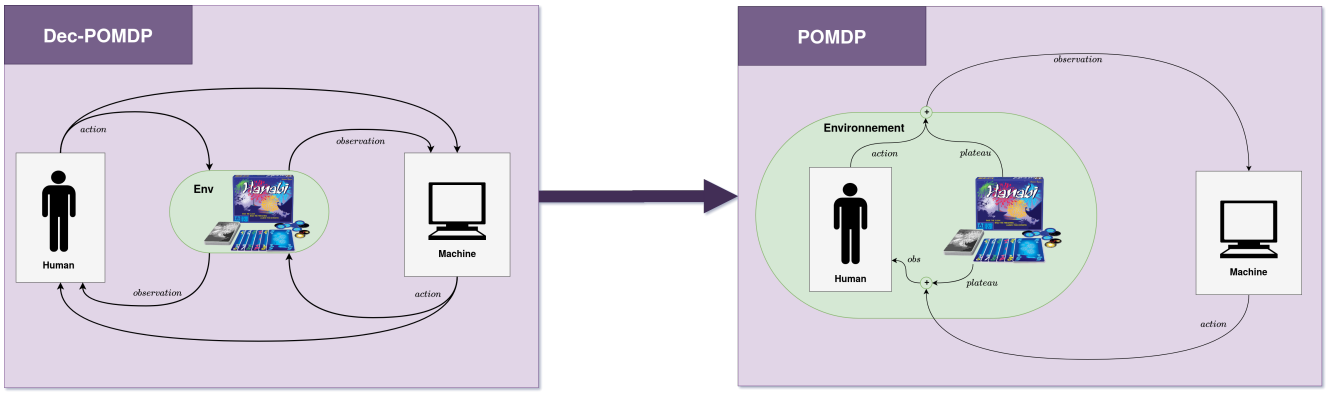


FIGURE I.2 – Passage Dec-POMDP à POMDP par changement de point de vue

3.2 Les recherches sur les systèmes de dialogue

Une autre communauté scientifique dont les recherches nous ont aidé est la communauté travaillant sur les systèmes de dialogue. En effet, le cadre d'étude d'un humain et d'une machine cherchant à collaborer par le dialogue semble proche de ce qui se fait dans le domaine.

Ces recherches nous ont principalement été utiles pour la première contribution (voir partie II) et nous ont aidé à :

- comprendre la gestion du dialogue
- formaliser le concept d'interlocuteurs collaboratifs
- comprendre les problématiques liées à une communication coûteuse
- connaître et garder en tête les applications "réelles" du problème

3.2.1 Gestion de dialogue et POMDP

Un système de dialogue est le méchanisme qui, lors d'une interaction avec un utilisateur, consiste à comprendre l'acte de dialogue de celui-ci et à formuler une réponse adéquate. Dans le cadre d'un

dialogue parlé, ce méchanisme comprend plusieurs étapes (schéma I.3). Tout d'abord, l'étape de reconnaissance automatique du speech permet de passer d'un enregistrement audio à une chaîne de caractères. Ensuite, la chaîne de caractères est interprétée pour pouvoir mettre à jour la croyance sur l'état du dialogue. Pour finir, un processus de décision sélectionne une réponse qui sera transformée en chaîne de caractères en langage naturel puis en une réponse audio.

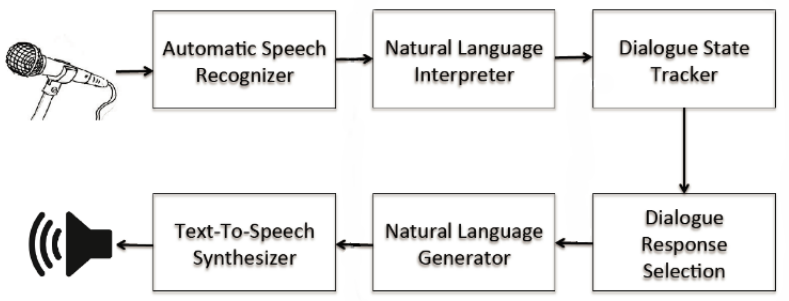


FIGURE I.3 – Méchanisme complet d'un système de dialogue parlé [8]

On remarque rapidement que les parties *Dialogue State Tracker* et *Dialogue Response Selection* se rapprochent des notions de croyance et de politique d'action vues dans la section 3.1.1. Une seconde manière d'appréhender notre problème serait donc de considérer que nous cherchons à élaborer un *Dialogue Manager* [4] dans le cadre d'Hanabi.

4. Le *Dialogue Manager* est le processus qui s'intéresse à la compréhension du dialogue et au contrôle de son avancement.

Partie II

Apprendre à jouer avec un coéquipier fixé

Apprendre à jouer de façon adéquate avec la plupart des humains est une tâche complexe. Cette tâche est d'autant plus difficile qu'il est difficile d'obtenir un grand nombre de données provenant de parties humaines. Dans cette partie, nous présentons les premières expérimentations réalisées dans le cas où l'agent doit s'adapter à un coéquipier fixé. Nous introduisons ensuite un modèle d'apprentissage des intentions permettant de décomposer le problème initial en sous-problèmes et d'accélerer l'apprentissage.

1 Création d'une stratégie pseudo-humaine

Pour apprendre à jouer de manière adéquate avec un coéquipier fixé, il a d'abord fallu concevoir ce coéquipier. L'objectif est que ce joueur utilise des règles proches de celles des humains. Ce coéquipier, que nous appellerons *SimpleBot-v1*, suit quatre règles avec un ordre précis. Il est fort probable que la première règle ne soit pas réalisable à un certain moment de la partie. Dans ce cas, SimpleBot-v1 jouera selon la règle suivante et ainsi de suite.

Priorité	Action
1	Jouer carte supposée jouable
2	Donner indice informatif
3	Défausser 1ère carte
4	Donner indice non informatif

FIGURE II.1 – Règles suivies par SimpleBot-v1

1.1 Evaluer une stratégie

Il n'est pas évident de trouver une métrique pour évaluer une politique d'action dans Hanabi. Dans nos expérimentations nous avons choisi la méthode suivante. Etant donné un coéquipier C, nous évaluons une politique P en comparant les résultats du duo (P, C) avec les résultat du duo (C, C). Cette métrique a du sens car, en général, les joueurs jouant selon la même logique performent mieux dans Hanabi. Si, par ailleurs, nous souhaitons tester la capacité d'un agent à jouer avec un grand nombre de coéquipiers, nous devrons avoir une mesure de performance qui prenne en compte le jeu avec un ensemble d'agents conséquent.

Ainsi, le score moyen réalisé par SimpleBot-v1 avec lui-même sur 1000 parties de jeu est égale à ***14.1***/25. Ce score est loin du score maximum mais permet de réaliser les premières expérimentations.

2 Apprendre à collaborer par essais/erreurs

Cette section présente les expérimentations réalisées dans le but d'apprendre à jouer avec un coéquipier fixé.

2.1 Simplified R2D2

Le premier modèle à avoir été utilisé est le modèle R2D2 présenté dans la partie I. Nous avons implémenté une version simplifiée de ce modèle en s'inspirant de l'implémentation du papier [9]. Nous avons ensuite testé notre algorithme d'apprentissage avec différentes configurations. Malgré de nombreuses tentatives, les résultats des expérimentations étaient loin d'être convainquants. Plusieurs hypothèses ont été émises sur les raisons de ces résultats. Tout d'abord, il est probable que la simplification du modèle ait joué sur la détérioration des performances de celui-ci. Une seconde hypothèse, peut-être la plus probable, est que nous ne disposions pas d'assez de ressources de calcul et qu'ainsi nous étions encore loin du point de convergence. Quelques résultats commentés sont disponibles en annexe B.1.

2.2 Rainbow DQN

Malgré tous les efforts, il a semblé nécessaire de revenir à un algorithme plus simple. Le choix s'est porté sur l'algorithme Rainbow DQN. Rainbow DQN [10] est une variante de l'algorithme DQN qui combine toutes les améliorations apportées entre 2015 et fin 2017 au Deep Q-learning. L'inconvénient de l'utilisation de cet algorithme est qu'il n'est pas propice aux environnements très partiellement observable comme Hanabi. Cependant, nous avons remarqué que la plateforme Hanabi fournit dans l'observation une grande partie des informations utiles depuis le début de la partie (voir annexe B.2). Il ne paraît donc pas aberrant de considérer l'observation à un instant t comme étant suffisante au choix d'une action presque optimale. A partir de maintenant, toutes les expérimentations utiliseront le modèle Rainbow DQN.

2.3 Expérimentations

Nous avons donc lancé une longue expérimentation sur *gollum*, une machine du laboratoire, dans le but d'apprendre à jouer avec SimpleBot-v1. Sur la figure II.2, nous voyons qu'après un peu plus d'un demi million de parties ($\approx 24h$), notre agent commence à dépasser les performances du bot en *self-play*. C'est seulement après avoir joué plus de deux millions de parties ($\approx 5\ jours$) avec le bot que la politique d'action de notre agent, nommé *RainbowDQN*, semble avoir convergé vers une politique quasi-optimale. L'équipe *(RainbowDQN, SimpleBot-v1)* réalise ainsi un score moyen d'environ ***17***/25, surpassant de près de trois points l'équipe *(SimpleBot-v1, SimpleBot-v1)*.

D'autres expérimentations ont été réalisées dans le but d'apprendre à jouer avec d'autres agents que SimpleBot-v1. Un exemple intéressant est la stratégie d'AwwBot. AwwBot est une IA pour Hanabi qui joue de façon non conventionnelle. Le style de jeu de cette IA est assez loin de ce que les humains ont tendance à faire. Cependant, l'étude de la capacité de Rainbow DQN à trouver une réponse qui soit bonne à des stratégies très différentes tels que les stratégies *Simple* et *Aww* sera intéressante par la suite.

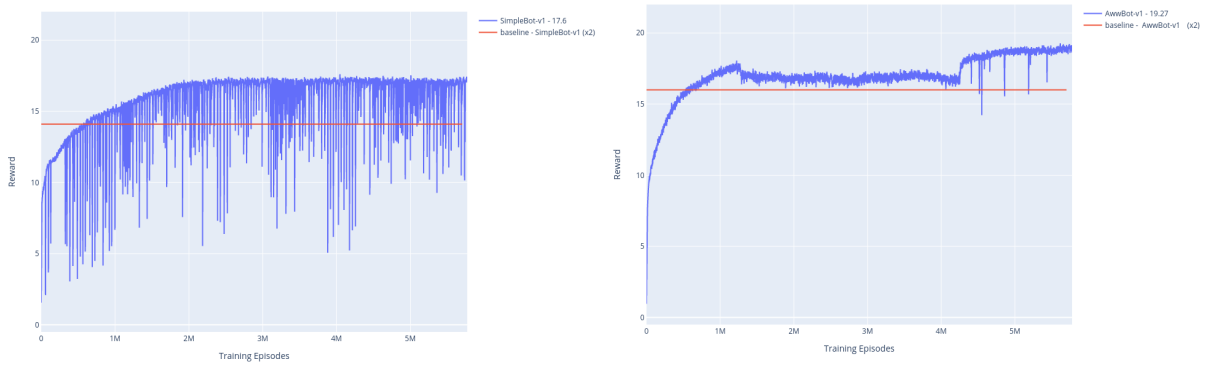


FIGURE II.2 – Courbe d'apprentissage suivant le score réalisé par l'agent en collaboration avec SimpleBot-v1 (gauche) et AwwBot-v1 (droite). La ligne rouge est la ligne de comparaison du bot en self-play.

Qualitativement, les résultats montrent que la méthode est appropriée à l'élaboration d'une bonne réponse pour un coéquipier fixé. En effet dans les deux exemples de la figure II.2 notre agent surpasse les scores du bot avec lui-même. Il est toutefois bon de remarquer que même dans ce cas assez simple, l'apprentissage reste coûteux en temps et en ressources de calculs. La section suivante présente une méthode visant à améliorer ces points.

3 Décomposer le processus

Notre première contribution a été réalisée avec la volonté d'accélérer l'apprentissage en décomposant la tâche principale en sous-tâches. Nous avons remarqué que, pour formuler une réponse convenable, l'agent doit :

1. Comprendre l'intention de l'utilisateur
2. Formuler une réponse en fonction de cette intention et du contexte

Dans les expérimentations précédentes, nous n'utilisions qu'un seul bloc pour faire tout cela. Cependant, cela rend le processus de décision fastidieux. Dans les études sur les systèmes de dialogues, la partie *compréhension de l'intention utilisateur* est souvent privilégiée. Prenons un exemple concret. Un usager aborde un chatbot SNCF en lui disait "Trois tickets pour Paris s'il te plaît.". En général, dans cette situation, le chatbot a appris à comprendre l'intention de l'utilisateur mais ne sait pas répondre directement. Il classifie donc l'intention utilisateur comme *buy tickets* avec comme informations supplémentaires *destination="Paris"* et *nombre=3*. Connaissant l'intention de l'utilisateur, une réponse est désormais plus simple à formuler. C'est sur cette idée que sont basées les expériences de cette section.

La classification des intentions utilisateur nécessite généralement des données labellisées difficiles à obtenir. Nous proposons donc dans cette partie un protocole d'apprentissage par renforcement multi-agents dans lequel un agent apprend à comprendre l'intention du coéquipier et l'autre agent apprend à réagir de façon adéquate à cette intention.

3.1 Comprendre l'intention utilisateur

Le premier module, que nous appelerons *module d'intention*, consiste donc à comprendre l'intention du coéquipier. Nous avons souhaité rester sur notre thématique d'apprentissage et éviter le plus possible d'utiliser des connaissances expertes. Pour cette raison, nous avons choisi de reformuler le problème de classification de l'intention qui, à priori, nécessite de labelliser des données en un problème d'apprentissage par renforcement. Ainsi, le but de l'agent consistera à interpréter au mieux l'acte de dialogue utilisateur d étant donné le contexte c. La qualité de l'interprétation i influera, par

le biais d'une transformation intention/action, sur la récompense r perçue par l'équipe (voir la reformulation du problème de classification de l'intention figure II.3). Dans le jeu Hanabi, si l'utilisateur donne un indice sur une de nos cartes, son intention, derrière cette action, est probablement de nous dire qu'il faut jouer cette carte. L'agent devra apprendre à classifier l'intention comme *"play"* et ainsi jouer cette carte.

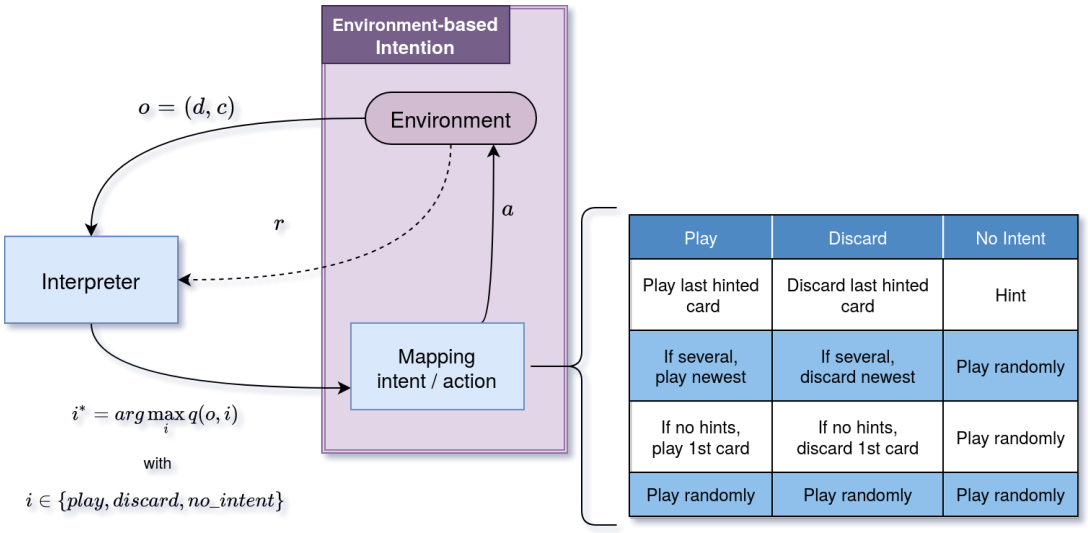


FIGURE II.3 – Boucle d'apprentissage de l'interprétation de l'intention (cas de trois intentions).

3.1.1 Expérimentations

Tout d'abord, nous avons expérimenté ce premier module seul. Dans ce cas, nous cherchons à répondre à différentes questions telles que :

— La solution de ce problème est-elle plus simple ?
— Quel est l'impact d'une augmentation du nombre d'intentions sur les résultats ?
— Permet-il d'accélérer d'une quelconque manière l'apprentissage ?

Pour éviter des interprétations erronées, lors des expériences ci-après, nous avons fixé tous les hyperparamètres du modèle sauf celui dont nous voulons tester l'influence.

Simplification du problème La partie interprétation est modélisée par un réseau de neurones. Une intuition que nous pouvons avoir est que l'interpréteur optimal de la figure II.3 est une fonction moins complexe que la politique d'action complète de l'agent. Ainsi, nous devrions pouvoir modéliser cette première fonction par un modèle de plus faible capacité.

En analysant les courbes de la figure II.4, nous pouvons confirmer notre intuition précédente. En effet, le passage de deux couches cachées de 512 neurones à deux couches de 256 neurones n'impacte pas les performances dans le cas du problème de classification d'intentions (courbes bleue et rouge), tandis que pour le problème original [1] (courbe verte), celles-ci sont diminuées. De plus, l'architecture à deux couches de 128 neurones semble être suffisante uniquement pour le module à trois intentions.

En conclusion, la formulation introduite a une solution plus simple qui permet de diminuer les ressources mémoires nécessaires. Cependant, cette simplification a un coût.

1. La boucle d'apprentissage originale peut être vue comme notre *module d'intention* dans lequel les intentions sont les actions possibles et le mapping *intention/action* est la fonction identité.

Impact de l'augmentation du nombre d'intentions Essayons désormais de comprendre les conséquences d'un nombre d'intentions plus grand sur les performances. Intuitivement, nous espérons que les résultats seront d'autant meilleurs que le nombre d'intentions est grand. En effet, plus le nombre d'intentions est grand et plus le mapping intention/action est simple, précis, et commun à tout type d'agent. Dans la figure II.4, nous voyons que le passage de trois intentions (courbe bleue) à onze intentions (courbe rouge) permet une augmentation des résultats de plus de deux points.

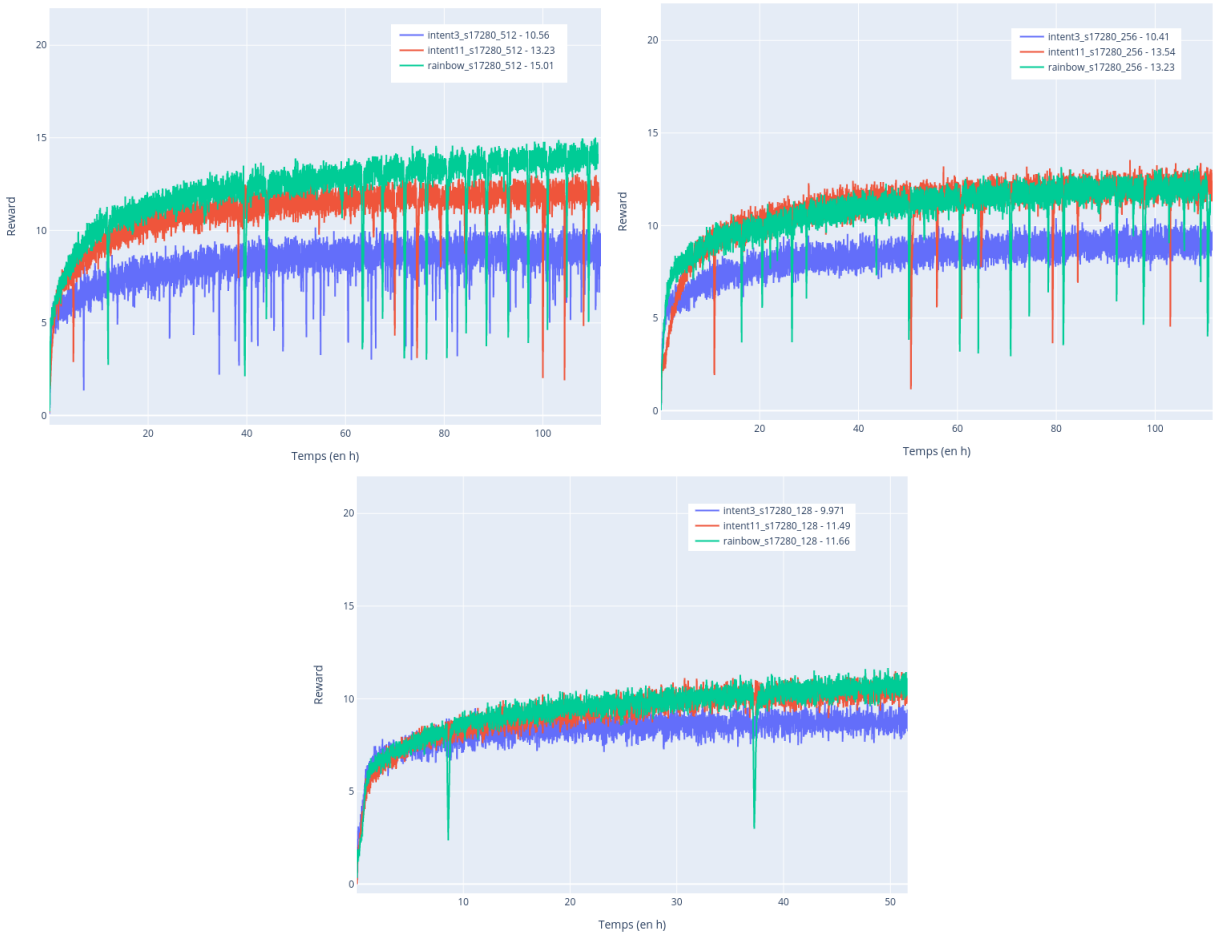


FIGURE II.4 – Courbes d'apprentissage suivant les scores moyens réalisés au cours du temps par trois modèles : modèle complet dit *Rainbow DQN*, modèle d'intention à 3 intentions et modèle d'intention à 11 intentions. Haut-Gauche : 512 neurones par couche. Haut-Droite : 256 neurones par couche. Bas : 128 neurones par couche.

Nous pouvons également nous demander s'il est pertinent d'augmenter considérablement le nombre d'intentions. La réponse est non. En augmentant le nombre d'intentions, nous risquons de converger vers le problème initial et donc de supprimer la simplicité de la solution du modèle introduit ici.

Ainsi, pour tirer les meilleurs bénéfices de ce modèle, nous devons avoir une transformation *intention/action* adéquate et trouver le bon compromis entre trop d'intentions et trop peu d'intentions.

3.2 Réagir à une intention utilisateur

Le second module, que nous appelerons *module de réaction*, utilise l'intention prédite et l'observation courante pour déterminer l'action à exécuter (figure II.5). L'intérêt de ce module est qu'il permet de ne plus reposer sur un transformation *intention/action* faite à la main. Pour se faire, nous augmentons l'observation de l'agent par le vecteur d'intention et ajoutons une pénalité $\lambda \in [0, 1]$ s'il ne suit pas

l'intention dont la valeur est la plus grande. Suivre les conseils du coéquipier sera donc vu comme une tâche auxiliaire à la tâche principale. Plus l'hyperparamètre λ est choisi grand et plus l'agent aura la nécessité d'écouter les conseils de son coéquipier pour ne pas être pénalisé.

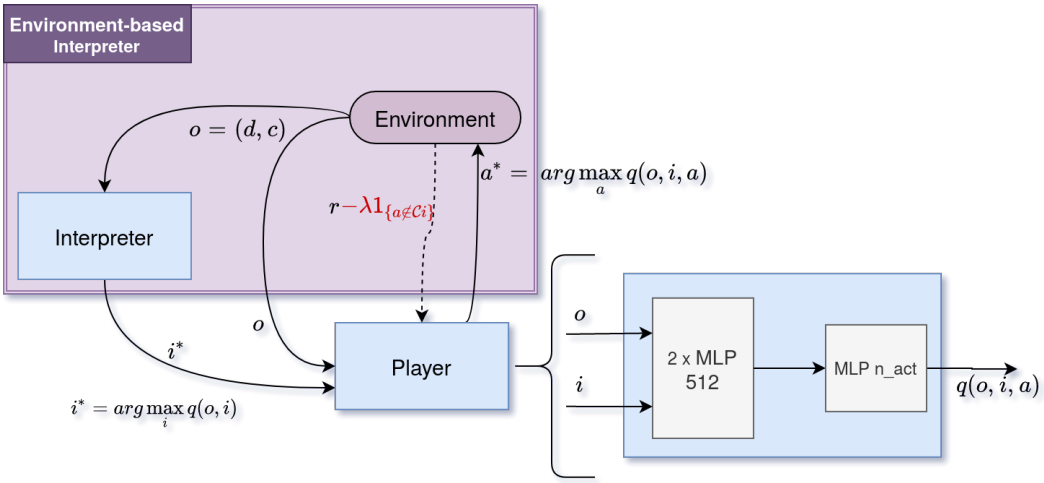


FIGURE II.5 – Boucle d'apprentissage avec tâche auxiliaire

Nous avons fixé le bloc d'interprétation et expérimenté le module de réaction en jouant sur le paramètre λ. La figure II.6 montre les résultats comparatifs des phases d'apprentissage avec le paramètre λ égale à 0.1 et 0.3. On remarque que plus λ est élevé et moins les résultats sont bons. Une interprétation possible serait de dire que l'agent suit aveuglément les conseils de son coéquipier, même quand ceux-ci ne sont pas pertinents.

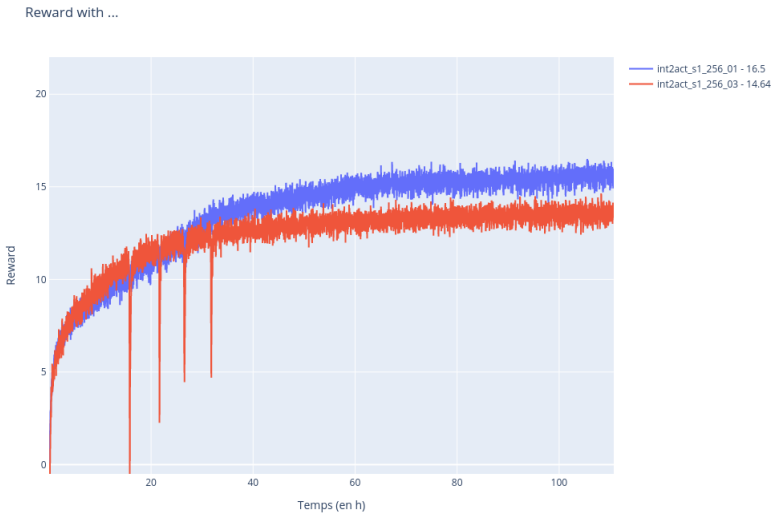


FIGURE II.6 – Courbes d'apprentissage suivant les scores moyens réalisés au cours du temps par deux modèles de réaction avec des paramètres λ différents.

3.3 Combiner les deux

Le modèle final que nous aurions souhaité expérimenter alterne les deux phases d'apprentissage précédentes pour construire la fonction d'interprétation et la réaction simultanément. Ce cas aurait permis de supprimer le "mapping" intention/action qui est source de résultats à la baisse. Malheureusement, par manque de temps, nous n'avons pas pu tester ce processus complet.

Partie III

Apprendre à jouer avec une population de coéquipiers

Jusqu'à présent, nous n'avons parlé que du cas d'un coéquipier unique. Dans cette partie nous nous intéressons à la problématique de jeu avec des coéquipiers différents. Nous présentons certaines des idées imaginées pour aborder le problème et les résultats d'expérimentations pour l'une d'entre elles.

1 Adaptation au profil utilisateur

1.1 L'idée

Une première idée qui avait été mentionnée consistait à s'adapter à des profils de joueurs bien définis. Plus précisément, on suppose qu'il y a environ n façons de jouer que l'on nomme **profils**. Chaque joueur peut être caractérisé par l'un de ces profils. Tout comme la séparation *intention/action* introduite dans la partie II, nous aurons cette fois une séparation *profil/action*. L'idée consiste à décomposer le problème en deux. Une partie classification du profil de l'utilisateur puis une partie qui utilise l'estimation du profil pour choisir l'action optimale. Un schéma détaillé de la structure du modèle imaginé est présenté en figure III.1.

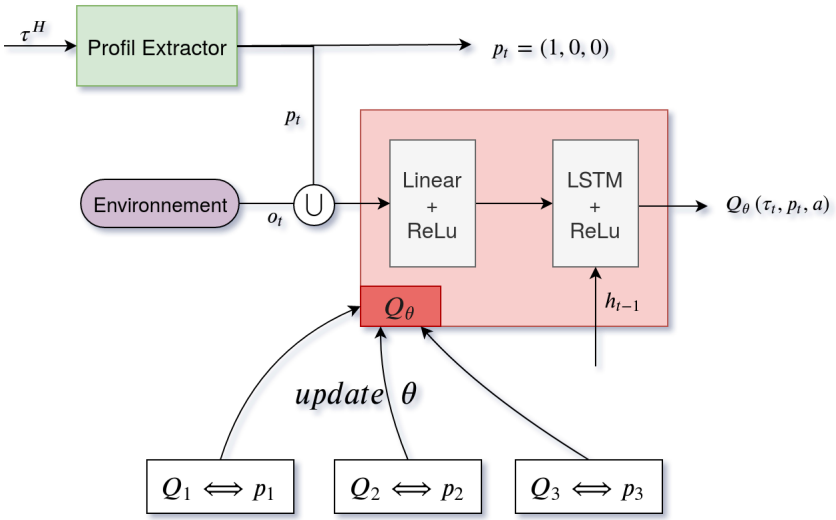


FIGURE III.1 – Modèle d'adaptation au profil utilisateur.

Dans ce modèle, nous ne spécifions pas comment construire l'*extracteur de profils* (en vert). Plusieurs possibilités s'offrent à nous. L'une d'entre elles serait de construire à la main un ensemble de règles

permettant de reconnaître le profil du joueur. L'inconvénient de cette solution est qu'elle nécessite d'avoir une connaissance experte sur les règles utilisées par les différents profils de joueur. L'autre solution consisterait à apprendre à classifier le profil du joueur. Nous pourrions faire cela de deux manières. Soit de façon supervisée (dans ce cas il faut simplement connaître le profil des coéquipiers lors de la phase d'entraînement), soit de façon non supervisée (dans ce cas nous pouvons utiliser le même type de procédure que lorsque nous avons parlé de classification des intentions).

1.2 Adaptation en ligne

En définitive ce modèle permettrait de jouer de manière presque optimale avec un ensemble fini de coéquipiers. Dans l'optique d'une collaboration avec l'homme il permettrait de fournir en instantané une première solution sous-optimale pour tout nouveau coéquipier. L'optimalité pourrait ensuite être obtenue en reprenant l'apprentissage à partir de cette première solution et en l'améliorant directement en ligne (c'est-à-dire pendant les parties avec ce nouveau coéquipier).

2 Apprendre un comportement moyen optimal

L'idée précédente est intéressante mais relativement complexe à mettre en place. De plus, nous avons remarqué précédemment qu'une architecture de réseau de neurones artificiels suffisamment complexe pouvait permettre d'encapsuler assez d'informations pour ne pas avoir besoin de deux phases d'apprentissage distinctes. Nous avons donc privilégié la méthode présentée ci-après pour la réalisation des expérimentations.

Dans cette seconde contribution nous cherchons à construire un joueur moyen. C'est-à-dire un joueur qui performe au mieux en moyenne avec tout le monde. Nous ne considérons plus le fait de déterminer le profil du coéquipier de manière explicite. A la place, nous espérons que le modèle d'approximation de la fonction de valeur q ait une capacité [1] suffisamment grande pour capter les caractéristiques de jeu du coéquipier grâce à l'historique.

2.1 Construire une politique d'action robuste aux changements de coéquipiers

Dans tout problème d'apprentissage, nous souhaitons que la fonction apprise généralise bien les données. Si c'est le cas, les performances sur de nouvelles données resteront bonnes. Dans le cas contraire, de faibles variations sur les données engendrent des résultats catastrophiques. C'est le cas de l'agent de la partie II qui a appris à jouer parfaitement avec un coéquipier précis (SimpleBot-v1) mais n'est bon qu'avec ce coéquipier.

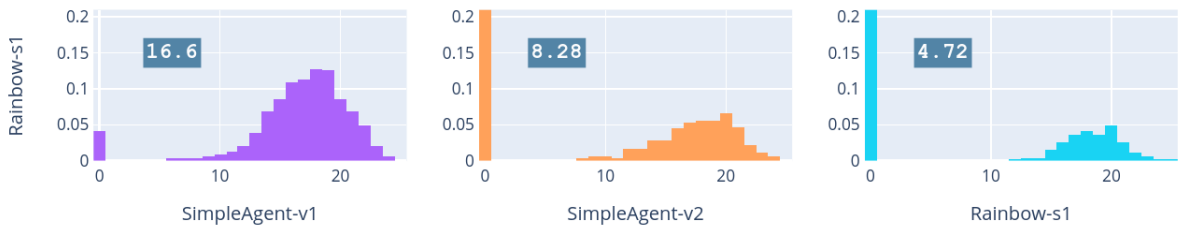


FIGURE III.2 – Distribution des scores obtenus pour chaque coalition (x, y) avec $x \in \{SimpleBot\_V1, SimpleBot\_V2, Rainbow\_S1\}$ et $y = Rainbow\_S1$. Chaque sous-figure représente la fréquence d'apparition des scores et le score moyen est en gris.

Dans la figure III.2 on voit que l'agent, nommé *Rainbow-s1*, réalise un score moyen de **8.3**/25 avec *SimpleBot-v2* contre **16.6**/25 avec *SimpleBot-v1*. Bien que la politique d'action de la version 2 n'ait que

1. La capacité d'un réseau de neurones artificiel représent l'étendue des fonctions que celui-ci peut approximer

peu changé, les performances deviennent catastrophiques. Dans ce cas, on parle de sur-apprentissage (ou *overfitting* en anglais). L'agent n'a pas su généraliser les données pour construire une stratégie robuste à de petites variations.

Pour tenter d'avoir un agent plus robuste à des changements de stratégies, nous devons le faire jouer avec un ensemble conséquent de joueurs pseudo-humains. Nous avons donc construit de nombreuses stratégies basées sur un ensemble de règles.

2.2 Simuler des stratégies "pseudo-humaines"

Rappelons que l'objectif initial est de collaborer avec des humains. De ce fait, toutes les stratégies simulées utilisent des règles qui pourraient être utilisées par les humains. A titre d'exemple, nous avons des règles telles que : défausser une carte aléatoire, jouer la dernière carte indicée, défausser la plus ancienne carte, donner un indice sur une carte jouable, etc. Dans cette partie nous nous préoccupons du problème de généralisation. Nous tirerons aléatoirement 90% des stratégies générées pour la phase d'entraînement. Les 10% de stratégies restantes seront utilisées pour tester la capacité du modèle à généraliser à d'autres coéquipiers.

Au total, plus de 115000 stratégies différentes ont été générées. Certaines stratégies utilisent une suite d'action extrêmement mauvaise du fait de la méthode de génération automatique expliquée en annexe C.1. Suffisament mauvaise pour que même les meilleurs coéquipiers ne puissent pas réaliser un score supérieur à 2/20. Nous avons donc sélectionné des sous-ensembles de joueurs dont les stratégies ont plus de sens.

Les trois sous ensembles utilisés pour les expérimentations sont les suivants :

- **Simple-4320** : ne comprend que des agents dont l'ordre des règles principales est :
  1. ***jouer (P)***
  2. ***donner un indice utile (U)***
  3. ***défausser (D)***
  4. ***donner un indice non nécessaire (N)***
- **Simple-17280** : comprend tous les agents dont l'ordre des règles est :
  1. soit ***P***, soit ***U***
  2. soit ***P***, soit ***U*** (celui qui n'est pas choisi en 1)
  3. soit ***D***, soit ***N***
  4. soit ***D***, soit ***N*** (celui qui n'est pas choisi en 3)
- **Simple-103680** : comprend tous les agents du générateur

2.3 Expérimentations

Pour chacun des trois ensembles d'agents (Simple-4320, Simple-17280 et Simple-103680), nous avons visualisé la courbe d'apprentissage mais aussi les performances avec les données de tests pour nous assurer de la capacité du modèle à généraliser.

Comme nous pouvons le constater sur la figure III.3, les courbes d'apprentissage semblent converger vers des scores moins élevés que ceux obtenus en apprenant à jouer avec un coéquipier fixe. Plusieurs raisons sont à l'origine de ce phénomène. La première raison vient du fait que les joueurs considérés sont moins bons. En effet, d'après la métrique du *self-play*, l'agent SimpleBot-v1 performe mieux en moyenne que ceux générés par le générateur (voir fig. III.4). Ainsi, même si la stratégie apprise est optimale, les scores réalisés seront inférieurs.

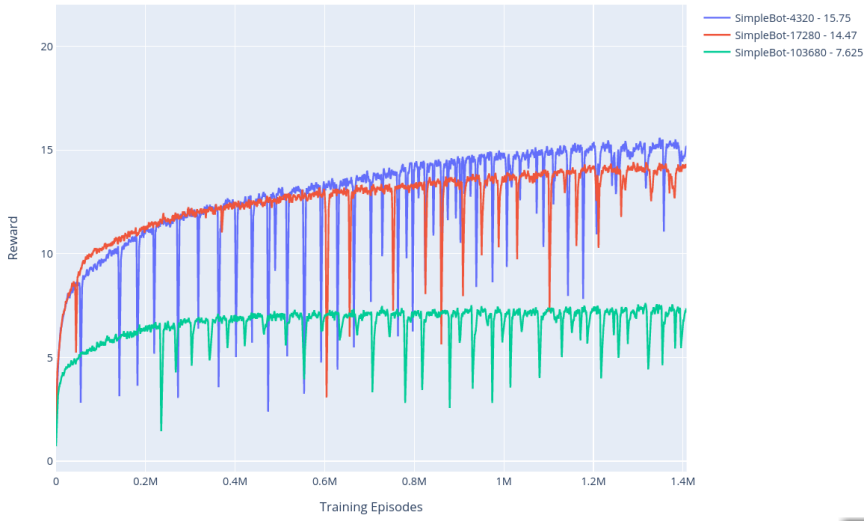


FIGURE III.3 – Courbes d'apprentissage suivant le score réalisé par l'agent en collaboration avec les agents *Simple-4320*, *Simple-17280* et *Simple-103680*

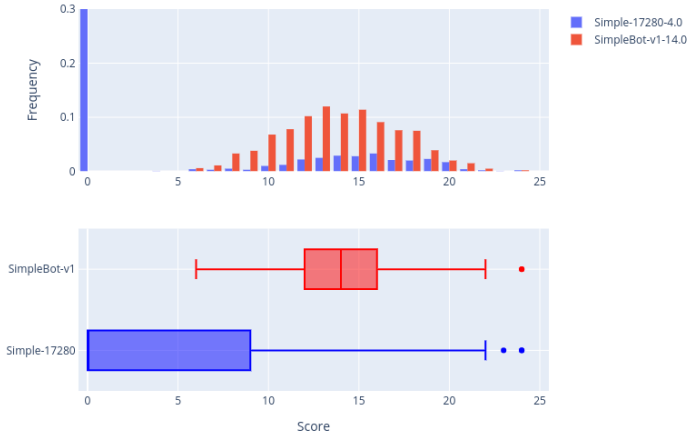


FIGURE III.4 – Résultats en self-play de SimpleBot-v1 et Simple-17280.

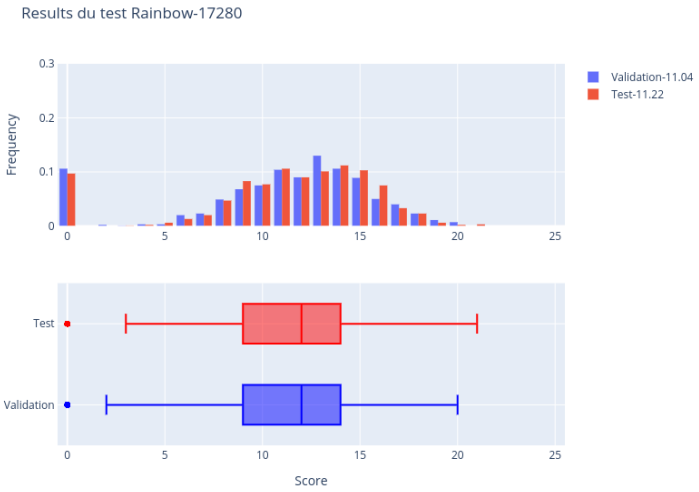


FIGURE III.5 – En bleu : Validation sur un échantillon de 1000 agents tirés parmis les 90% d'apprentissage. En rouge : Test sur un échantillon de 1000 agents tirés parmis les 10% de test.

La seconde raison de cette baisse de performance vient du fait que notre algorithme apprend à construire un joueur moyen et non un joueur optimal pour chaque coéquipier. Cela signifie que pour une même observation de l'environnement, lors de parties avec des joueurs différents, nous ne pourrons choisir qu'une seule action à exécuter. En conclusion, l'agent apprendra à jouer une action bonne pour tout le monde mais optimale pour personne, ce qui peut affecter les résultats. Heureusement, nous avons remarqué que, en général, plus les agents sont différents et moins il est probable d'avoir des observations identiques (voir annexe C.2).

Nous avons ensuite analysé la capacité du modèle à généraliser et à jouer avec des coéquipiers nouveaux. Les résultats de Rainbow-17280 avec les 10% d'agents réservés pour le test montrent que le modèle a réussi à généraliser. En effet, les performances avec les agents de test ne semblent pas dégradées par rapport aux performances avec les agents d'apprentissage (voir figure III.5). Nous obtenons des résultats similaires avec Rainbow-4320 et Rainbow-103680.

Partie IV

Conclusion

1 Perspectives du stage

De nombreuses perspectives seraient à envisager pour le projet. En priorité, nous trouverions intéressant d'expérimenter les modèles présentés sur de vraies données humaines.

1.1 Utiliser des données humaines

La contrainte de simulation d'un joueur humain est forte. Il aurait été intéressant de relâcher cette contrainte en utilisant de vraies données humaines. Dans le cas où nous souhaiterions n'utiliser QUE des parties humaines, il aurait été trop difficile de récolter suffisamment de données. Cependant, un entre-deux est une piste à envisager pour des travaux futurs. Il consisterait à utiliser dans un premier temps le travail réalisé durant ce stage pour obtenir une première stratégie robuste à différents joueurs. Ensuite, via une récolte *en live* de données de parties d'humains avec notre agent, ce dernier pourrait continuer à progresser au fur et à mesure de parties réellement humaines. Ce travail nécessiterait la réalisation d'une plateforme en ligne pour récolter les données.

1.2 Repasser à un modèle récurrent

Une seconde perspective serait de tenter de repasser à l'utilisation d'une architecture de réseau de neurones récurrent. Malgré la difficulté à mettre en place, les RNN nous permettraient sans doute de garder plus d'informations pour caractériser le coéquipier que l'observation seule. Nous pourrions ainsi envisager des performances optimales pour des coéquipiers très différents.

2 Conclusion personnelle

Ce stage de fin d'étude était pour moi une première expérience en laboratoire de recherche. J'ai eu la chance de travailler sur un sujet très intéressant et tout à fait d'actualité avec deux enseignants-chercheurs renommés. Initialement, la problématique globale ne m'avait pas paru aussi large et diversifiée qu'elle l'est. Ce n'est qu'après avoir entamé quelques recherches et expérimentations que j'ai compris l'étendue du travail nécessaire pour pouvoir couvrir tous les aspects du problème. De plus, la partie expérimentale en *Deep Learning* est conséquente et prend du temps sur le travail un peu plus théorique. J'ai donc pu me perfectionner sur les aspects d'analyse de résultats expérimentaux qui est essentiel pour ce travail. J'ai toutefois été un peu déçu que les contributions n'aient permis que de démystifier certains points du problème et non d'apporter une réelle contribution qui aurait pu prendre la forme d'une publication scientifique.

Bibliographie

[1] N. Bard and al. The hanabi challenge : A new frontier for ai research. *Artificial Intelligence Journal*, 2019.

[2] DeepMind. Plateforme hanabi, 2019.

[3] OpenAI. Openai gym.

[4] D. Albert. Projet de fin d'études : Apprentissage par renforcement multi-agents. 2020.

[5] Mnih et al. Human-level control through deep reinforcement learning. 2015.

[6] Hausknecht and Stone. Deep recurrent q-learning for partially observable mdps. 2017.

[7] Kapturowsk et al. Recurrent experience replay in distributed reinforcement learning. 2019.

[8] I. Serban et al. A survey of available corpora for building data-driven dialogue systems. 2017.

[9] Hu and Foerster. Simplified action decoder for deep multi-agent reinforcement learning. 2019.

[10] Hessel et al. Rainbow : Combining improvements in deep reinforcement learning. 2018.

Partie V

Annexes

A Annexes Partie 1

A.1 Comparaison des algorithmes existants

Regime	Agent	2P	3P	4P	5P
–	SmartBot	22.99 (0.00) 29.6%	23.12 (0.00) 13.8%	22.19 (0.00) 2.076%	20.25 (0.00) 0.0043%
–	FireFlower	22.56 (0.00) 52.6%	21.05 (0.01) 40.2%	21.78 (0.01) 26.5%	-
–	HatBot	–	–	–	22.56 (0.06) 14.7 %
–	WTFWThat	19.45 (0.03) 0.28%	**24.20** (0.01) **49.1%**	**24.83** (0.01) **87.2%**	**24.89** (0.00) **91.5%**
SL	Rainbow	20.64 (0.11) 2.5 %	18.71 (0.10) 0.2%	18.00 (0.09) 0 %	15.26 (0.09) 0 %
UL	ACHA	22.73 (0.12) 15.1%	20.24 (0.15) 1.1%	21.57 (0.12) 2.4%	16.80 (0.13) 0%
UL	BAD	**23.92** (0.01) **58.6%**	–	–	–

FIGURE V.1 – Tableau de performances

A.2 Définition *Dec-POMDP*

Definition A.1. ***Dec-POMDP***
Le DEC-POMDP est décrit par un tuple $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ avec :

- $\mathcal{S}$: espace d'états
- $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times ... \times \mathcal{A}_n$: ensemble d'espaces d'actions propre à chaque agent
- $\mathcal{T} : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to [0, 1]$: fonction de transition de l'état s à l'état s' en exécutant l'action jointe $\boldsymbol{a}$
  $\to \mathcal{T}(s, \boldsymbol{a}, s') = \mathbb{P}(S_{t+1} = s' \,|S_t = s, A_t^{(1)} = a_1, ..., A_t^{(n)} = a_n) = p(s' \mid s, a_1, ..., a_n)$
- $\mathcal{R} : \mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \times \mathbb{R} \to [0, 1]$: récompense probabiliste liée au choix de l'action jointe $\boldsymbol{a}$
- $\boldsymbol{\Omega} = \Omega_1 \times ... \times \Omega_n$: ensemble d'espaces d'observations propre à chaque agent
- $\mathcal{O} : \Omega \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \to [0, 1]$: fonction d'observation probabiliste
  $\to \mathcal{O}(o, \boldsymbol{a}, s') = \mathbb{P}(O_{t+1} = o \,|A_t = a, S_{t+1} = s') = p(o \mid a, s')$

A.3 Algorithme du Q-learning

La règle de mise à jour de q dans l'algorithme du Q-learning est la suivante :

$$q_{t+1}(s, a) = q_t(S_t, A_t) + \alpha_t(\, \mathbf{R_{t+1}} + \gamma \max_{\mathbf{a'}} \mathbf{q_t}(\mathbf{S_{t+1}}, \mathbf{a'}) - q_t(S_t, A_t))$$

La fonction q_π étant une moyenne probabiliste du gain G_t, la règle ci-dessus n'est autre qu'une moyenne mobile sur les $R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a')$ (estimateur biaisé de G_t). Sous certaines conditions sur le paramètre α_t, on a $q_t \underset{t\to\infty}{\longrightarrow} q_*$. L'avantage du Q-learning par rapport à d'autres méthodes tels que *TD-learning* est qu'il permet d'estimer directement q_* et non q_π. Dans le cadre d'un MDP fini, la politique d'action optimale sera ainsi donnée par $\pi(a \mid s) = \mathbb{1}_{a = arg\max_{a\in\mathcal{A}} q_*(s,a)}$.

A.4 Algorithme R2D2

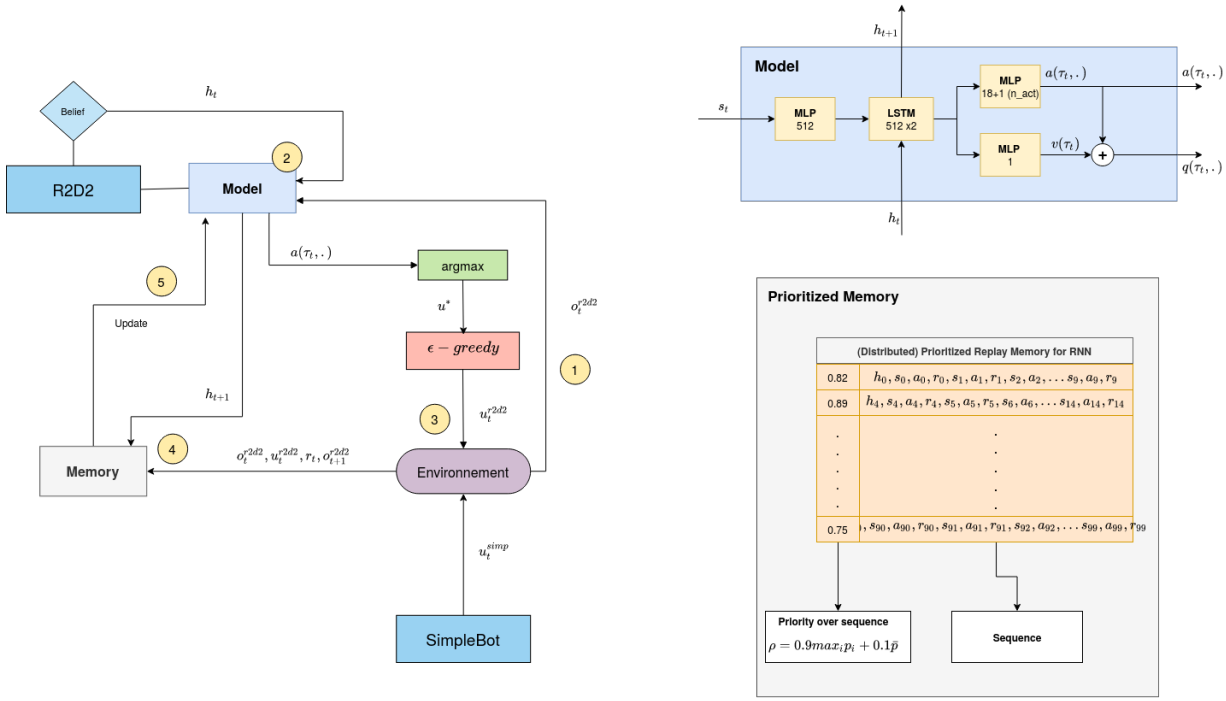


FIGURE V.2 – Schéma explicatif de l'algorithme R2D2

A.5 Classification des méthodes

Technique	Algorithmes	Description
Agents Indépendants	• IQL IAC - Independent Q-learning/Actor-Critic • PHC - Learning to cooperate via Policy Search • Coopération passive - MARL : Independent vs Cooperative	Les agents apprennent "chacun" dans leur coin. Suppose en général que les agents ont des tâches suffisamment indépendantes.
Partage de politiques	• PS-TRPO • Cas 2 - MARL : Independent vs Cooperative	Chaque agent utilise une politique d'action partagée qu'il mets à jour grâce à son expérience locale. Dans ce cas, les agents doivent être homogène.
Partage d'informations entre agents	• Cas 1 - MARL : Independent vs Cooperative • Coopération active - MARL : Independent vs Cooperative	L'observation des agents contient des informations sur les autres agents. Un canal de communication doit être mis en place.
Modélise l'adversaire ou le coéquipier	• JAL - Joint Action Learners • DRON - Opponent Modeling in Deep Reinforcement Learning	Modélise le joueur en face pour mieux s'adapter à lui.
Améliore la politique en fixant celle des autres	• JESP - Joint Equilibrium Search for Policies • SPARTA single-agent - Improving Policies via Search in Cooperative POSG	Si toutes les politiques sont fixées sauf une, alors l'environnement est stationnaire du point de vue de l'agent apprenant. Les méthodes classiques peuvent ainsi être utilisées pour trouver une solution sous-optimale.
Un dieu dicte des informations aux autres	• BAD - Bayesian Action Decoder • MADDPG - Multi-Agent Actor Critic for mixed Cooperative-Competitive Environments • COMA - Counterfactual Multi-Agent Policy Gradients • MS-MARL - Master-Slave Multi-Agent RL	Durant la phase d'apprentissage, les agents apprennent à se coordonner grâce à une entité qui a une vision plus gloabale et qui leur dicte des informations. Au moment du test, le groupe a appris à se coordonner et exécute la politique jointe apprise sans l'aide du 'dieu'.

FIGURE V.3 – Classification de méthodes

B Annexes Partie 2

B.1 Résultats du modèle R2D2

Les tests sur le modèle R2D2 ont été longs. Un très grand nombre de configurations différentes ont été testées. L'architecture du modèle, le processus d'exploration, la taille du buffer d'expérience et le taux d'apprentissage sont des exemples de paramètres qui ont été modifiés et testés.

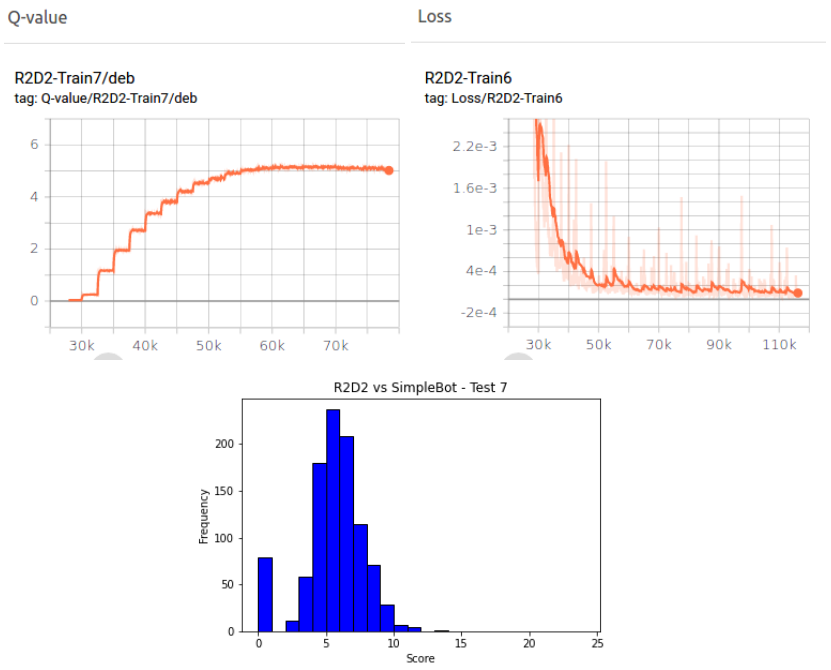


FIGURE V.4 – Exemple de q-value, loss et résultats de test obtenus avec R2D2.

B.2 Description de l'observation

La plateforme Hanabi utilisée [2] met à disposition de l'agent un vecteur de 783 valeurs réelles contenant de nombreuses informations sur l'état du jeu. Il sagit en réalité d'un résumé de toutes les observations perçues jusqu'à l'instant t.

Observation à un instant t :

1. Life tokens : 2
2. Info tokens : 7
3. Fireworks : R1 Y1 G0 W1 B0

Hands :

Cur player

XX	Y2 | Y2
XX	XX | RGWB1345
XX	XX | RYGWB1345
XX	XX | RYGWB1345
XX	X2 | RYGWB2

(6)

—

B5	XX | RYGWB12345
Y3	XX | RYGWB12345
B1	XX | RYGWB12345
Y4	XX | RYGWB12345
G4	XX | RYGWB12345

(5) (4)

7. Deck size : 31
7. Discards : W3 W2 W4 W4 G4 W3

Dans la vue humaine de l'observation ci-contre, de nombreux éléments sont disponibles. A savoir :

1. Le nombre de jetons de vies restantes
2. Le nombre de jetons d'indices restants
3. L'avancée des feux d'artifices
4. Les cartes du coéquipier
5. Les connaissances qu'a le coéquipier sur ses cartes
6. Les connaissances que j'ai sur mes cartes
7. La taille de la pile et les cartes défaussées

C Annexes Partie 3

C.1 Méthode de génération des agents

La génération des bots se fait de la façon suivante. Chaque bot utilise une méthode de mise à jour des croyances parmis deux différents. Ensuite, les agents sont dotés d'une permutation aléatoire de quatre actions génériques :

— Jouer une carte (P)
— Défausser une carte (D)
— Donner un indice utile (U)
— Donner un indice non nécessaire (N)

Les agents peuvent avoir une manière différente d'exécuter ces quatres actions. Par exemple, pour l'action D='défausser une carte', un certain agent défaussera la plus ancienne carte tandis qu'un autre agent défaussera une carte choisi aléatoirement. En tout, nous avons implémenté cinq façons d'exécuter l'action P, six pour l'action D, 1 pour l'action U et huit pour l'action N.

A la construction des agents ci-dessus nous avons rajouté une dernière caractéristique ayant pour objectif de simuler le fait qu'en général les humains ne suivent pas un ordre bien précis de règles. Le principe est que chaque agent a un *taux de maladresse* compris entre 0 et 0.2. Ce taux de maladresse représente la probabilité de ne pas suivre l'ordre des actions. Le taux de maladresse maximal (à savoir 0.2) a été déterminé en analysant l'impact de celui-ci sur la dégradation des scores pour un agent donné (voir fig. V.5).

La combinaison de toutes ses possibilités a permis de générer un nombre conséquent d'agents différents (≈ 115000).

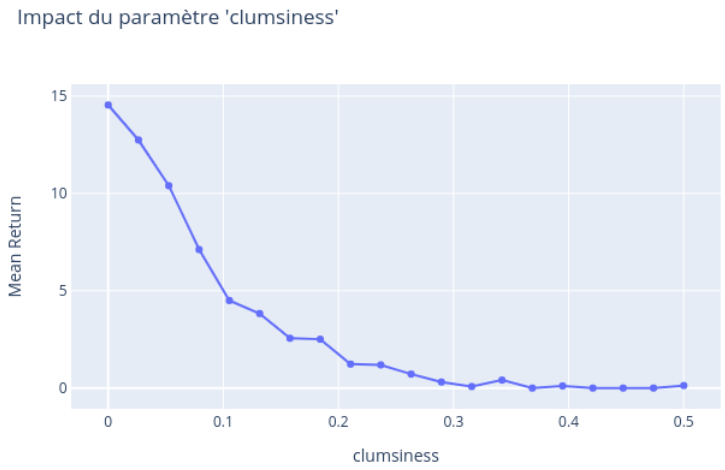


FIGURE V.5 – Impact du paramètre de maladresse sur les résultats

C.2 Stratégie moyenne avec des coéquipiers très différents

Nous avons constaté un point intéressant concernant le fait d'apprendre une stratégie moyenne pour des coéquipiers très différents. Avant de réaliser les expérimentations, il semblait qu'il serait difficile de construire une politique d'action qui soit capable de collaborer à la fois avec SimpleBot-v1 **et** avec AwwBot. En effet, nous pouvions supposer que l'agent apprendrait à ne pas prendre de risque en évitant de donner des indices qui ne seraient pas compris de la même façon par les deux joueurs. Or, sans indice, les résultats sont généralement très mauvais. A notre grande surprise, ça n'a pas été le cas. L'agent a appris à donner les bons indices en fonction du joueur en face. Comment cela est-il possible ? Cela est possible car le fait de jouer avec AwwBot génère un historique qu'il est peu probable de percevoir lors de parties avec SimpleBot. Ainsi, l'historique et dans notre cas l'observation (moins robuste mais fonctionne dans cet exemple) suffit à reconnaître notre coéquipier. La figure ci-dessous montre les résultats obtenus lors de la phase de test avec les deux agents.

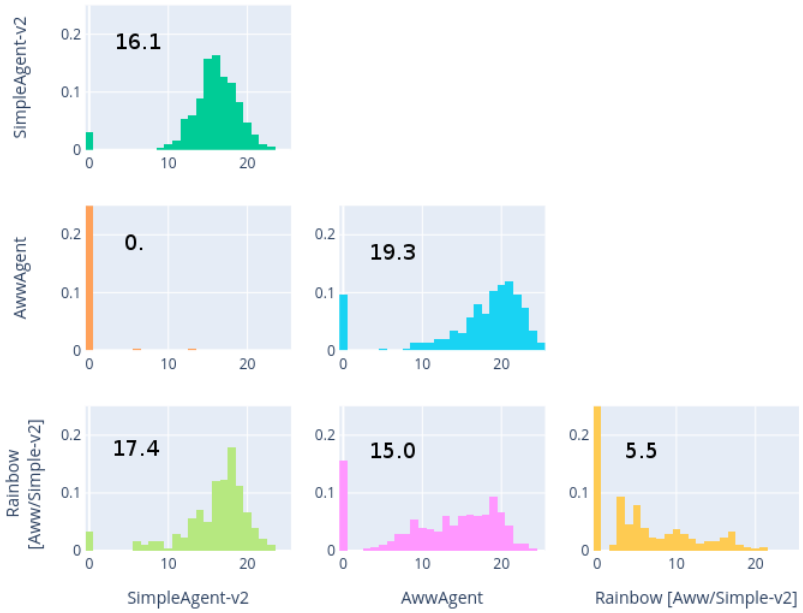


FIGURE V.6 – Distribution des scores obtenus pour chaque coalition (x, y) avec $x, y \in \{SimpleBot\_V2, AwwBot, Rainbow\_A1S1\}$.