Université de Montréal

Architecture et programme d'entraînement pour agents qui apprennent par renforcement

par

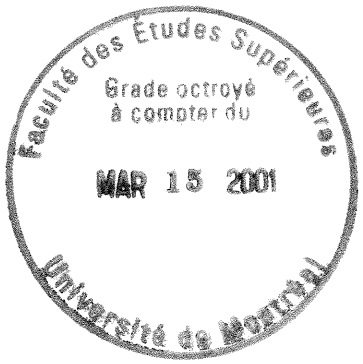
Julien Desaulniers

Département d'informatique et recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)

Août 2000



Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Architecture et programme d'entraînement pour agents qui apprennent par renforcement

présenté par:

Julien Desaulniers

a été évalué par un jury composé des personnes suivantes:

Esma Aïmeur	présidente du jury
Yoshua Bengio	directeur de recherche
Michel Gendreau	codirecteur de recherche
Victor Ostromoukhov	membre du jury

Mémoire accepté le : _____

Table des matières

Liste des tableaux

Table des figures

Sommaire

Ce projet a pour but de concevoir des agents intelligents capables d'apprentissage et d'établir les étapes à suivre pour leur faire apprendre une tâche spécifique. Le domaine d'application visé ici est le divertissement électronique, aussi communément appelé jeu vidéo.

Les acteurs de ce domaine démontrent un intérêt croissant pour les nouvelles techniques d'intelligence artificielle. Bien que certains travaux concernant l'application de ces techniques à ce domaine sont réalisés dans le milieu industriel, très peu le sont dans le milieu académique. Comme les travaux entrepris en milieu industriel sont pratiquement toujours de nature confidentielle, la mise en oeuvre de recherches en milieu académique est d'un certain intérêt. D'une part, les résultats de ces travaux sont accessibles à tous. D'autre part, ces travaux sont effectués avec des considérations plus scientifiques, moins économiques.

Le choix de l'algorithme d'apprentissage s'est rapidement orienté vers la famille des algorithmes d'apprentissage par renforcement. Ces algorithmes permettent aux agents d'apprendre une tâche avec un minimum de supervision. La seule information qui est fournie à l'agent en plus de l'information concernant l'état actuel de la scène est une appréciation de sa dernière décision. La solution qui a été retenue pour l'approximation des fonctions qui régissent ces algorithmes est l'utilisation de réseaux de neurones artificiels.

Une étude comparative de différents algorithmes de cette famille a déterminé que l'algorithme de descente de gradient sur la politique est le mieux adapté à la tâche étudiée ici. L'utilisation d'une couche cachée dans les différents réseaux de neurones n'apporte aucun gain. Des différentes techniques d'entraînement mises à l'essai, la plus efficace et la plus simple consiste à entraîner un groupe d'agents en parallèle et de conserver l'agent obtenant les meilleures performances suite à cette phase d'apprentissage.

La première partie de cet ouvrage explique d'avantage le contexte dans lequel le projet s'est déroulé. Ensuite, un bref aperçu de l'histoire de l'utilisation de l'intelligence artificielle dans le divertissement électronique est présenté. Par la suite, le cadre mathématique théorique sur lequel repose les recherches présentées ici est donné. Suit alors la description complète de la tâche apprise par les agents. L'architecture développée pour le système de prise de décisions est ensuite expliquée. Le module de planification de trajectoire est indépendant du reste du système de prise de décisions. Il fait l'objet

de la section suivante. Se trouve ensuite une section qui retrace l'ensemble des démarches expérimentales. Finalement, on retrouve une conclusion qui résume les résultats obtenus et indique les améliorations que l'on pourrait apporter au système et les directions dans lesquelles les futures recherches pourraient être poussées.

Remerciements

Je remercie M. Yoshua Bengio, directeur, et M. Michel Gendreau, codirecteur, des nombreuses suggestions qu'ils ont formulées tout au court du projet. Leur intérêt pour mon travail et leur contribution à celui-ci sont d'une valeur inestimable.

Je remercie également M. François Gilbert, directeur artistique, M. Yan Cyr, président, et M. Patrick Buisson, directeur de projet, de Enzyme Digital Marketing qui ont mis en oeuvre ce projet et qui m'ont permis de travailler sur celui-ci.

Je tiens de plus à souligner l'apport de M. Eric Fournier, directeur général, et M. Thierry Carle, programmeur en chef, de Insane Logics, par leurs nombreuses suggestions et remarques pertinentes émises lors de conversations informelles.

Finalement, je remercie mes parents, sans qui rien de tout ceci n'aurait été possible.

Chapitre 1

Introduction

Au cours des dix dernières années, le champ de l'apprentissage par renforcement [92] a connu une formidable croissance. L'effort consacré à la recherche dans ce domaine est sans cesse grandissant. Les résultats théoriques obtenus sont de plus en plus satisfaisants. Les techniques développées permettent de mettre sur pied des systèmes qui sont à la fois robustes et flexibles sans nécessiter des coûts de développement prohibitifs. C'est pourquoi l'industrie privée a commencé à manifester un intérêt pour ce domaine né du mariage de l'intelligence artificielle et de la psychologie cognitive. Bien qu'elles commencent à apparaître, les applications pratiques utilisant les techniques d'apprentissage par renforcement sont encore peu nombreuses. En plus de susciter l'intérêt des entreprises privées, les applications pratiques permettent de valider les concepts théoriques. Elles viennent démontrer la viabilité des avancements réalisés en milieu académique.

La nature de l'apprentissage par renforcement lui concède une grande flexibilité qui lui permet donc d'adresser un grand nombre de problèmes. Dans le modèle habituel, l'agent qui apprend par renforcement fait face à une situation pour laquelle il dispose d'un certain nombre d'informations. L'agent doit prendre une série de décisions dans le but d'accomplir la tâche qui lui a été attribuée. Après chaque décision, l'agent reçoit les informations les plus récentes concernant l'état du monde ainsi qu'une évaluation de la dernière décision lui indique la souhaitabilité de cette celle-ci. Le système apprend en cherchant à optimiser l'appréciation reçue.

Les travaux de recherches rapportés dans ce mémoire visent à mettre en application les principes d'apprentissage par renforcement, de paire avec les techniques de réseaux de neurones artificiels, dans un contexte industriel pour un problème spécifique. Le domaine d'application qui nous intéresse ici est celui du divertissement électronique ou plus communément appelé

jeux vidéo. L'objectif est l'entière réalisation des agents d'un jeu et de leur programme d'entraînement. Il faut donc développer les algorithmes d'apprentissage les mieux adaptés, bâtir une architecture de réseaux de neurones permettant un apprentissage optimal et établir un processus d'entraînement qui soit suffisamment court pour permettre une certaine flexibilité tout en demeurant assez complet pour obtenir des agents offrant des performances satisfaisantes.

1.1 Collaboration avec l'industrie

On a vu au cours des dernières années se resserrer encore davantage les liens déjà forts entre le milieu académique et le milieu industriel. Ceci est non seulement vrai pour l'intelligence artificielle ou même l'informatique en général, mais également pour un grand nombre d'autres sphères d'activités. Bien qu'elle ne soit pas sans susciter de débats [23][86], cette relation de collaboration profite largement aux deux parties. Les avantages que retire de cette relation le milieu académique ne sont pas sans importance. D'une part, comme il a déjà été mentionné, le contexte industriel offre la possibilité de réaliser des applications qui viennent valider les travaux de recherches plus fondamentales. En ce sens, le travail effectué en milieu industriel vient compléter les travaux déjà en cours dans le milieu académique. Certains auteurs défendent avec vigueur l'importance de la tenue d'expériences pratiques en science informatique [100]. D'autre part, pour ce qui est de considérations plus terre-à-terre, les fonds de recherche qu'injecte le milieu industriel dans le milieu académique permettent à ce dernier de poursuivre et accroître ses activités de recherches. Par contre, cette union réduit dans une certaine mesure la liberté d'action des chercheurs puisque les investisseurs privés tiennent évidemment à avoir leur mot à dire dans l'orientation des travaux qu'ils financent.

À l'inverse, cette relation apporte également des bénéfices aux entreprises. En finançant la recherche scientifique, les entreprises peuvent obtenir des produits qui bénéficient des derniers progrès scientifiques à des coûts comparables et parfois même inférieurs à ce qu'il en coûterait pour développer un produit répondant aux mêmes exigences soit exclusivement à l'interne de l'entreprise, soit par les services de sous-traitants. Ensuite, les entreprises s'assurent de la pertinence de la formation reçue par les futurs chercheurs. Ces derniers font déjà face, au cours de leurs études, à des problèmes très près de ceux auxquels ils seront confrontés dans le milieu professionnel.

1.2 L'apprentissage par renforcement

L'utilisation des techniques d'apprentissage par renforcement pour le développement d'agents intelligents dans les jeux vidéo ouvre un grand nombre de possibilités. Tout d'abord, au cours des dernières années la complexité des comportements a connu une très forte croissance empruntés par les agents dans les jeux. Il s'agit d'un marché très concurrentiel où chaque avantage technologique, aussi minime soit-il, revêt la plus grande importance. Dans ce contexte, le développement de ces agents se fait de plus en plus tôt dans le cycle de vie du logiciel, par opposition au passé, où il était fréquent d'amorcer le développement de ces agents dans les semaines qui précédaient la mise en marché d'un produit [110]. Par conséquent, il arrive souvent, de nos jours, que le reste du système n'est pas opérationnel ou du moins pas encore totalement au moment où on doit entreprendre le développement des agents. Conséquence de l'utilisation des techniques d'apprentissage par renforcement, une partie de la conception nécessaire est maintenant beaucoup moins spécifique au jeu développé que l'utilisation de techniques plus classiques, comme la conception d'une heuristique utilisant par exemple un ensemble de règles SI-ALORS. Il en résulte une meilleure réutilisation du code.

Ensuite, l'utilisation des techniques d'apprentissage par renforcement simplifie beaucoup le travail d'analyse. Plutôt que de tenter de mettre sur pied des stratégies dont l'efficacité n'est aucunement garantie puisqu'elles ne pourront être testées que beaucoup plus tard, le travail consiste à mettre sur pied une architecture ouverte pour permettre aux agents de trouver eux-mêmes quelle sera la stratégie optimale. Ceci a pour effet d'obtenir une meilleure flexibilité pour le travail d'analyse effectué. Ainsi, si des changements mineurs sont apportés au concept du jeu, les impacts de ceux-ci sur l'analyse seront d'une importance minimale.

Finalement, l'utilisation des techniques d'apprentissage par renforcement a des conséquences bénéfiques même du point de vue de l'usager. Bien qu'une phase d'apprentissage ait généralement lieu par simulation dans les bureaux du développeur, il est envisageable et très souhaitable que l'apprentissage se poursuive chez l'usager. Ainsi, les agents continueront d'améliorer la qualité de leur jeu au fur et à mesure que l'usager s'améliore lui-même. Les agents pourront même adapter leurs stratégies au style de jeu de l'usager. De plus, selon les joueurs et selon les jeux, certains usagers pourront prendre plaisir à jouer le rôle de l'entraîneur et chercher à obtenir les agents les plus performants, les comparer avec ceux d'autres joueurs et se les échanger, via Internet par exemple.

Le prochain chapitre est en quelque sorte une petite revue bibliogra-

phique de la littérature scientifique portant sur l'intelligence artificielle appliquée aux jeux. Le chapitre 3 présente le cadre mathématique de la théorie des algorithmes d'apprentissage par renforcement ainsi qu'une brève revue bibliographique de ce domaine. Le chapitre 4 résume le concept et les règlements du jeu qui nous intéresse. Le chapitre 5 donne un aperçu de l'architecture du système développé. La planification de trajectoire est un élément important de la conception d'agent pour les jeux 3D. Le chapitre 6 porte sur les solutions utilisées face à ce problème. Le chapitre 7 présente l'ensemble de la démarche scientifique qui a mené à l'obtention des agents et de leur programme d'entraînement. Finalement, le chapitre 8 conclut et suggère des travaux futurs.

Chapitre 2

L'IA et les jeux

Depuis le jour où il a inventé l'ordinateur, l'homme a toujours tenté de lui apprendre à jouer à des jeux. D'une part, l'homme a toujours eu besoin de divertissement. L'ordinateur possède plusieurs qualités qui font de lui un adversaire idéal : il est toujours disponible, son niveau de compétence peut théoriquement être variable et réglable et de plus, à moins qu'il ne reçoive des instructions contraires, il ne triche jamais. D'autre part, l'apprentissage de différents jeux est une partie importante du processus d'apprentissage de tout être humain. À cet égard, si l'humanité veut en venir un jour a créer des ordinateurs "*intelligents*", il est impensable que ce cheminement ne passe pas, à un moment ou un autre, par le jeu.

Au fil des années, la complexité des jeux que l'on a appris aux ordinateurs a évolué avec la capacité de calcul de ceux-ci. Les techniques employées ont changé dramatiquement depuis le premier programme de Tic-Tac-Toe. Les jeux de nos jours offrent aux joueurs de l'action en temps-réel et des agents aux comportements de plus en plus complexes. Les joueurs s'attendent à ce que les adversaires commandés par IA soient une source de défis et de surprises. Nous proposons donc ici un aperçu des travaux de recherche qui ont été réalisés dans ce domaine ainsi que l'état actuel de la science. Une attention particulière est portée aux travaux concernant l'apprentissage par renforcement.

2.1 Espaces d'états et d'actions discrets

Depuis les 50 dernières années, plusieurs jeux se sont mérités l'attention des chercheurs. Au départ, la recherche se concentrait typiquement sur des jeux dont l'espace des états possibles était discret, fini et, de préférence, relativement petit. Les "jeux de table" étaient donc grandement favorisés. Cela

était dû évidemment aux ressources limitées des machines de l'époque. On cherchait donc des jeux simples à représenter, autant pour des considérations d'affichage, de structures de données que de pure algorithmie. En 1950, Shannon publiait l'une des premières contributions au problème du jeu d'échecs [82]. Les échecs demeurent un des problèmes les plus classiques en intelligence artificielle. Encore de nos jours, il se fait beaucoup de recherches sur ce sujet. Certaines publications scientifiques se consacrent uniquement à la publication d'articles de recherches sur l'intelligence artificielle appliquée aux échecs. C'est notamment le cas de *International Computer Chess Association Journal* et de *Advances in Computer Chess.* Ces dernières années, certains chercheurs se sont tournés vers les algorithmes d'apprentissage par renforcement pour tenter de trouver de nouvelles avenues à la recherche dans ce domaine [22] [37]. Par contre, les travaux qui ont de loin retenu le plus d'attention sont ceux réalisés par le groupe de recherche de IBM avec sa série *Deep Blue.* L'attention du public a été attirée non seulement par la qualité des résultats obtenus, mais aussi par la notoriété des noms impliqués (IBM, Kasparov) [50][49][47].

Le jeu de dames a également suscité l'intérêt des chercheurs. En effet, bien que ce jeu soit un peu plus simple que celui des échecs, il n'en demeure pas moins un défi intéressant. Dès 1959, Samuel publie l'un des premiers travaux dans le domaine des dames [80]. Depuis, un grand nombre de jeux ont été étudiés. Certains jeux simples n'ont pas reçu beaucoup d'attention jusqu'à récemment pour diverses raisons. C'est notamment le cas du backgammon. En effet, non seulement l'espace des états possibles d'un jeu de backgammon est énorme (30 pièces avec chacune 26 emplacements possibles), mais ce jeu comprend aussi l'utilisation de dés. Les dés ajoutent un élément stochastique complexe à intégrer, particulièrement lorsqu'on utilise les techniques classiques de recherche et d'heuristique [1]. On peut également considérer la configuration des dés comme une variable d'état, mais cependant ceci vient accroître encore davantage l'espace des états possibles. En 1994, Tesauro a implanté un joueur de backgammon utilisant l'apprentissage par renforcement qui a atteint un niveau de jeu que l'on peut classifier comme *expert* [95][96]. Cependant, non seulement ce joueur de backgammon joue-t-il de façon remarquable, mais il a développé des stratégies qui sont aujourd'hui utilisées par des experts, particulièrement pour ce qui est des premiers coups. Typiquement, un joueur expérimenté connaît, pour le tout premier coup de la partie, pour chacune des configurations possibles des dés, le meilleur coup possible. Ces stratégies d'ouverture sont en quelque sorte une convention entre les joueurs. Le joueur de Tesauro a appris des stratégies d'ouverture différentes de celles qui étaient utilisées auparavant. Ces stratégies se sont avérées supérieures et sont maintenant utilisées par la majorité des joueurs de niveau expert [96]. Plusieurs autres jeux de table ont été étudiés. Les jeux Othello et Tic-Tac-Toe servent souvent d'exemples pour

démontrer de nouveaux algorithmes étant donné la simplicité des règles et des stratégies [1]. Un jeu plus récent, mais tout aussi simple, sert également régulièrement d'exemple. *Tetris*, bien que relativement nouveau, est connu de pratiquement tous et est utilisé de plus en plus à des fins de démonstration [26]. Bien que sa nature temps-réel le démarque des autres jeux mentionnés précédemment, le caractère hautement discret des espaces des états et des actions en font un exemple simple. Le jeu de go demeure un sujet très populaire grâce aux subtilités stratégiques qu'il permet. Bon nombre de chercheurs ont d'ailleurs tenté d'appliquer les techniques d'apprentissage par renforcement à ce problème [72][39].

Les jeux de cartes ont toujours été et demeurent encore un favori des chercheurs. On trouve bon nombre de publications sur les jeux les plus populaires comme par exemple le poker [27], le blackjack [74][107], le bridge [33] [34] et les coeurs [71][58]. Le nombre de cartes possibles étant connu, fini et relativement restreint, les problèmes de jeux de cartes sont généralement d'une complexité aisément traitable. Cependant, pour des jeux comme le poker, une grande partie du succès d'un joueur repose sur son habileté à anticiper la stratégie de son ou de ses adversaires. Évaluer rapidement le profil d'un joueur pour en extraire sa politique de jeu reste un problème intéressant [27]. Évidemment, pour l'instant l'ordinateur est tenu à l'écart du côté plus "humain" des cartes, c'est-à-dire celui de tenter de deviner les cartes que possède l'adversaire par ses expressions corporelles et de cacher son propre jeu en évitant d'émettre quelconque émotion. Mais on peut entrevoir un jour de pousser la simulation jusque là.

2.2 Espaces d'états et d'actions continus

Parallèlement, les jeux impliquant des espaces d'états et d'actions continus se sont mérités leur part d'attention. Dès le début des années 60, on voit apparaître des articles sur les jeux de poursuites : *Tag*, *Capturer le drapeau*, etc [51][84][83][97]. Ces jeux intéressent notamment le milieu militaire qui n'hésite pas à financer la recherche dans ce domaine [51]. Le problème classique des jeux de poursuite est sans contredit le problème du *chauffard meurtrier (homocidal chauffeur)* [51][83]. Dans ce problème, on a M chauffards et N fuyards dans un grand espace libre d'obstacles. L'objectif des chauffards est de frapper les fuyards alors que l'objectif de ces derniers est d'éviter la collision fatale.

Les sports d'équipe offrent d'excellents problèmes de recherche. En plus d'avoir à traiter des espaces d'états et d'actions continus, les agents doivent être en mesure d'opérer de façon conflictuelle avec certains agents, mais coopérative avec d'autres. Le soccer est un sport parmi les plus étudiés.

Sport par excellence à l'échelle planétaire, ce jeu mobilise à lui seul des centaines de chercheurs de par le monde [60]. Chaque année, des équipes de scientifiques de partout se rencontrent pour un tournoi amical d'agents de soccer. La compétition se déroule en deux catégories principales. Dans la première, le jeu se déroule par simulation sur une plate-forme standard pour laquelle les participants n'ont qu'à fournir le code qui sert au contrôle des agents. Dans la seconde catégorie, les agents sont de véritables robots et l'action est tout ce qu'il y a de plus réel. Cette compétition se subdivise davantage en d'autres catégories selon la taille et le poids des robots.

Plusieurs autres jeux, notamment des jeux moins classiques que ceux mentionnés précédemment, font également l'objet de recherches. Winstead s'est intéressé aux machines à boules (*pinball*). Il a développé un système qui apprend ce jeu [108]. Son système, se basant sur un algorithme d'apprentissage par renforcement, utilise une plate-forme de simulation qui offre déjà une discrétisation du temps et de l'espace. Le jeu *SimCity* a aussi été le sujet de différents travaux de recherches [41][40]. Dans ce jeu de simulation, le joueur doit veiller au bon fonctionnement d'une ville, jouant les rôles, tour à tour, de maire, d'urbaniste, de chef de police, etc. Bien que ce jeu soit discret par sa nature (l'espace de la ville est discrétisé par une grille), le nombre de possibilités étant si grand, il est très difficile de prendre avantage de cette nature discrète.

2.3 L'IA dans les jeux aujourd'hui

De nos jours, les attentes de la part des usagers face aux agents de jeux vidéo sont énormes. La puissance de calcul disponible sur les ordinateurs personnels grandit sans cesse. Une grande partie du traitement nécessaire à l'exécution d'un jeu, soit l'affichage, est maintenant prise en charge par du matériel dédié [110]. Les usagers veulent d'une part des comportements très complexes, surprenants et variés. D'autre part, les usagers veulent être en mesure d'éditer eux-mêmes les comportements adoptés par les agents et créer de nouveaux types d'agents [110][111].

En réponse à la première attente, plusieurs technologies ont été mises à l'essai avec des niveaux de succès variables : logique floue, algorithmes génétiques, réseaux de neurones. Cependant, une technologie qui est apparue ces dernières années retient l'attention. Il s'agit des techniques de Vie Artificielle (*Artificial Life*) [44]. Ces techniques, qui suggèrent une approche du bas vers le haut, visent à modéliser un système bio-mécanique complet. Plutôt que d'imposer une série de comportements, on indique à l'agent ses possibilités et ses besoins, et on laisse les comportements émerger d'eux-mêmes. Ces techniques sont à la base du jeu *Creatures*, dans lequel le joueur

doit mettre au monde une créature, l'élever et l'amener à la procréation dans le but final de constituer toute une communauté de ces créatures [45].

Mais les joueurs veulent également être en mesure de visualiser et d'éditer la logique qui régit le comportement des agents. Plusieurs jeux récents offrent ces possibilités au moyen d'un langage de script dédié à cette tâche. Comme on veut permettre aux usagers de comprendre ce langage, il doit être simple et intuitif. Son utilisation ne doit pas nécéssiter de notions de programmations approfondies. C'est notamment le cas des jeux *Quake* et *Unreal*. C'est une pratique courante dans l'industrie que de mettre sur pied un tel système pour faciliter le développement des agents à l'interne par les membres de l'équipe de développement. Ainsi, l'édition des comportements des agents ne relève plus exclusivement de l'équipe de programmation, mais peut également être prise en charge par l'équipe de conception artistique. Davis [36] présente l'architecture de la solution qu'a mise sur pied le développeur américain *Activision* pour le jeu *Planetfall 2 : The Other Side of Floyd*. Ce problème relève davantage de la conception de langage, d'analyse syntaxique et sémantique que d'intelligence artificielle. Le lecteur intéressé peut consulter [3] à ce sujet.

2.4 L'IA et le divertissement

Il ne faut pas perdre de vue que l'intégration de l'intelligence artificielle aux jeux vidéo a pour but premier d'offrir une expérience plus divertissante.

Dans cette optique, le module d'intelligence artificielle d'un jeu doit posséder certaines caractéristiques. En tout premier lieu, les performances des agents doivent être réglables sur plusieurs paliers. Ces différents paliers correspondent aux niveaux de difficultés offerts au joueur. Ensuite, si l'agent triche d'une quelconque façon, le joueur ne doit pas être en mesure de s'en apercevoir. L'agent peut avoir accès à de l'information sur des éléments en dehors de son champ de vision ou encore avoir des attributs nettement supérieurs à ceux du joueur. (L'agent pourrait par exemple se déplacer plus rapidement que le joueur).

Finalement, le scénario idéal d'un combat opposant un agent à un joueur humain prévoit, dans un premier temps, la domination apparente de l'agent suivie d'une séquence où le joueur humain échappe de justesse à la mort, le tout culminant par le triomphe de l'être humain sur l'agent. Ce type de scénario valorise beaucoup le joueur humain qui y prend alors grand plaisir. C'est à ce moment que l'expérience devient du divertissement.

Ces comportements complèxes requièrent souvent des techniques et des algorithmes sophistiqués. Le chapitre suivant présente les concepts utilisés

dans le cadre de ce projet.

Chapitre 3

L'apprentissage par renforcement

Ce chapitre présente le concept de base sous-jacent à l'ensemble des travaux rapportés ici, l'apprentissage par renforcement. D'abord, nous commençons par présenter une vue d'ensemble du concept. Ensuite, nous introduisons le cadre mathématique formalisant ce concept. Nous présentons ensuite une première idée d'où et comment s'intègrent les réseaux de neurones. Enfin, nous terminons par une revue bibliographique de la littérature pertinente au sujet.

3.1 Concept général

L'apprentissage par renforcement est une branche de l'intelligence artificielle qui a beaucoup gagné en popularité au cours des dix dernières années. Il s'agit d'une famille d'algorithmes situés à mi-chemin entre les algorithmes d'apprentissage supervisé et les algorithmes d'apprentissage non supervisé. Contrairement aux algorithmes d'apprentissage supervisé de classification qui reçoivent l'information concernant la classe d'une donnée, l'algorithme reçoit, comme information complémentaire à l'information à traiter, une valeur numérique lui indiquant le niveau de désirabilité de la dernière décision prise. C'est cette valeur numérique qui est appelée le signal de renforcement.

L'apprentissage s'effectue un peu comme le dressage d'un animal. Lorsque celui-ci adopte un comportement jugé souhaitable, il reçoit une récompense de la part du dresseur. Qu'il s'agisse de nourriture, d'une caresse ou simplement d'un compliment verbal, la récompense aura pour effet de renforcer le choix du comportement dans une situation semblable par la suite. C'est ce

que l'on appelle renforcement positif. À l'inverse, lorsque l'animal présente un comportement jugé indésirable, le dresseur le punit. Encore une fois, la punition peut prendre plusieurs formes : réprimande verbale ou physique, ou encore privation d'un plaisir. Si la punition est appliquée dans un délai assez bref après le comportement jugé indésirable, l'animal peut alors faire l'association et éviter de reproduire le comportement dans une situation semblable par la suite. C'est ce que l'on appelle renforcement négatif.

Les algorithmes d'apprentissage par renforcement apprennent en suivant le même modèle à une seule exception près : les renforcements sont purement numériques. Une récompense est représentée par une valeur positive alors qu'une punition est représentée par une valeur numérique négative. La valeur absolue du signal de renforcement indique l'importance de la récompense ou de la punition. De plus, on cherche à donner à ces algorithmes une vision à long terme. On veut que l'agent soit en mesure d'optimiser les récompenses obtenues pour la durée totale d'un épisode et non pour un seul pas de temps.

Ce concept intuitif est connu de tous. Voici maintenant comment il se formalise mathématiquement.

3.2 Cadre mathématique

Les tâches destinées aux techniques d'apprentissage par renforcement sont généralement formulées en termes de MDP (*Markov Decision Process*, ou Processus de décisions de Markov) [13][32] [76]. Un MDP est composé de trois éléments de base distincts : S, un signal d'état, A, un ensemble d'actions et $P^a_{ss'}$, les probabilités de transitions d'un état à un autre selon l'action choisie telles que définies par

$$P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{3.1}$$

où s_t est la valeur courante de la variable S au temps t et a_t est la valeur choisie pour la variable A au temps t.

À chaque pas de temps, un signal contenant de l'information sur l'état actuel de l'environnement est présenté à l'agent. Celui-ci choisit une action a sur la base de ce signal. Une fois que l'état de l'environnement a été modifié par cette action (et éventuellement d'autres facteurs extérieurs), un nouveau signal d'état est présenté à l'agent et le processus recommence de nouveau. Ceci suggère des pas de temps discrets, mais des généralisations existent pour traiter le temps de façon continue [16][94].

Pour que le problème soit caractérisé par la propriété de Markov, le

signal d'état doit résumer entièrement l'historique des observations passées. Les considérations relatives à l'état actuel sont indépendantes du chemin pris pour en arriver à cet état. Cette propriété peut être formulée comme suit :

$$Pr\{s_{t+1} = s' | s_t, a_t\} = Pr\{s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0\} \quad (3.2)$$

Le formalisme mathématique exprimant l'apprentissage par renforcement ajoute un élément supplémentaire au MDP de base. Après chaque action, l'agent reçoit un signal de renforcement. Ce signal agit comme une récompense ou punition numérique indiquant la qualité d'une décision. La valeur de ce signal, qui peut être stochastique, dépend du dernier état, de la dernière action et de l'état actuel.

$$R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \quad (3.3)$$

où r_{t+1} est la valeur du signal de renforcement au temps t+1.

3.2.1 Différence temporelle

L'algorithme TD (*Temporal Difference*) [13],[91], [21], offre une bonne alternative à la programmation dynamique pour résoudre les MDP quand les fonctions $P^a_{ss'}$ et $R^a_{ss'}$ régissant la dynamique de l'environnement ne sont pas connues. Les fonctions à apprendre sont la fonction d'évaluation des états

$$V^\pi(s) = E^\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \quad (3.4)$$

où $V^\pi(s)$, aussi noté simplement V, est l'espérance de la valeur cumulative des renforcements futurs en partant de l'état s si la politique π est suivie jusqu'à la fin de l'épisode et la fonction d'évaluation des actions

$$Q^\pi(s,a) = E^\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \quad (3.5)$$

où $Q^\pi(s,a)$, aussi noté simplement Q, est l'espérance de la valeur cumulative des renforcements futurs en choisissant l'action a dans l'état s si la politique π est suivie jusqu'à la fin de l'épisode. Le facteur d'atténuation γ,

dont la valeur est toujours située entre 0 et 1, a pour effet que les valeurs du signal de renforcement plus rapprochées dans le temps ont plus d'impact que les valeurs qui viendront plus tard dans le futur.

L'évaluation d'un état ou d'une action est en fait l'espérance mathématique du signal de renforcement à recevoir multipliée par un facteur d'atténuation,γ. Afin de calculer l'approximation la valeur des récompenses futures, TD utilise le *bootstrapping*. La valeur de l'état suivant, ou de l'action suivante, est utilisée comme estimé des récompense futures. Les équations (3.4) et (3.5) deviennent

$$V^{\pi}(s) = E^{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\} \tag{3.6}$$

et

$$Q^{\pi}(s,a) = E^{\pi}\{r_{t+1} + \gamma argmax_{a_{t+1}}(Q^{\pi}(s_{t+1}, a_{t+1})) | s_t = s, a_t = a\} \tag{3.7}$$

On suppose pour cette dernière équation que la politique suivie consiste a toujours choisir l'option la plus prometteuse.

En fait, une seule des ces deux fonctions est nécessaire pour être en mesure de prendre des décisions. La fonction Q^{π} permet de savoir directement quelle action maximise l'espérance des récompenses futures. Pour utiliser la fonction V^{π} afin de prendre une décision, il faut également connaître la fonction $P^{a}_{ss'}$. On peut ainsi savoir l'état qui résulterait de chaque option disponible. En évaluant la valeur de chacun de ces états, on peut finalement déterminer l'espérance des récompenses futures pour chaque option.

Il est indispensable de prendre en considération l'ensemble des récompenses futures et non seulement la récompense octroyée au pas suivant car, dans un bon nombre de tâches, les conséquences d'une décision peuvent être observées après des délais variables. Il s'agit alors d'une situation d'apprentissage par renforcement avec retard. C'est d'ailleurs le cas pour notre tâche.

3.2.2 Traces d'éligibilité

Dans un grand nombre de tâches, le choix d'une option peut avoir des conséquences selon une échelle temporelle variable. C'est pourquoi il est important d'avoir un mécanisme pour effectuer le décompte des états visités et des actions choisies. Ainsi, on est en mesure d'assigner le crédit pour le renforcement reçu à chaque pas aux états ou aux actions qui ont conduit

à l'obtention de ce renforcement. Les traces d'éligibilité constituent l'outil parfait pour répondre à ce besoin.

Dans sa forme de base, une trace d'éligibilité est conservée pour chaque état possible, notée $e_t(s)$. À chaque itération, la trace de l'état courant est incrémentée et les traces des autres états sont diminuées d'un facteur d'atténuation, γ. Nous avons donc

$$e_t(s) = \gamma e_{t-1}(s) \quad si\ s \neq s_t \tag{3.8}$$

et

$$e_t(s) = \gamma e_{t-1}(s) + 1 \quad si\ s = s_t \tag{3.9}$$

Chaque passage à chaque état est donc comptabilisé de façon cumulative. Ainsi, si un état est visité très fréquemment, ce haut niveau d'achalandage sera reflété par la trace d'éligibilité correspondant à cet état. Comme l'indique l'équation 3.6, la sortie désirée pour la fonction $V(s_t)$ est $V(s_t) = r_{t+1} + \gamma V(s_{t+1})$. À chaque pas, l'erreur est calculée selon l'équation

$$\delta_t = -\frac{\partial}{\partial V(s_t)} \frac{1}{2}[V(s_t) - (r_{t+1} + \gamma V(s_{t+1}))]^2 \tag{3.10}$$

donc

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{3.11}$$

L'apprentissage réalisé est calculé par

$$\Delta V(s) = \alpha \delta_t e_t(s) \tag{3.12}$$

où α est le pas d'apprentissage. L'apprentissage réalisé à chaque pas est limité car ce qui est vrai pour un cas ne se généralise pas nécessairement pour tous les autres. En limitant l'apprentissage effectué à chaque pas, seules les tendances qui se présentent à plusieurs reprises auront un impact.

Pour les algorithmes que nous utiliserons, les traces pour les couples action-état, notées $e_t(s, a)$, seront mieux adaptées. Nous obtenons donc, pour les équations régissants les traces d'éligibilité

$$e_t(s, a) = \gamma e_{t-1}(s, a) + 1 \quad si\ s = s_t\ et\ a = a_t \tag{3.13}$$

et

$$e_t(s,a) = \gamma e_{t-1}(s,a) \quad sinon. \tag{3.14}$$

L'erreur devient

$$\delta_t = r_{t+1} + \gamma argmax_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{3.15}$$

et la règle d'apprentissage devient

$$\Delta Q(s,a) = \alpha \delta_t e_t(s,a) \tag{3.16}$$

Traces remplaçantes

Si un état est souvent visité ou si la même action est souvent choisie pour un même état, la valeur des traces d'éligibilité peuvent rapidement grossir, ce qui entraînera une importance accrue des modifications apportées aux paramètres appris. Ceci peut finalement amener tout le système à s'emballer et à devenir hautement instable. Le système oscille alors entre deux ou plusieurs tendance sans jamais converger. Les valeurs des paramètres appris tendent vers $\pm\infty$. Finalement les valeurs de ces paramètres dépassent les limites de ce que peut supporter l'implantation de l'algorithme, ce qui provoque une interruption brusque de l'exécution. Les traces remplaçantes sont une solution à ce problème.

Plutôt que d'incrémenter de un la trace de l'état ou du couple état-action visité à chaque pas, on remplace la valeur actuelle de cette trace par un. On obtient donc les équations

$$e_t(s) = \gamma e_{t-1}(s) \quad si\ s \neq s_t \tag{3.17}$$

et

$$e_t(s) = 1 \quad si\ s = s_t \tag{3.18}$$

si les traces sont seulement spécifiques à l'état et

$$e_t(s,a) = 1 \quad si\ s = s_t\ et\ a = a_t \tag{3.19}$$

et

$$e_t(s,a) = \gamma e_{t-1}(s,a) \quad sinon \tag{3.20}$$

si les traces représentent des couples état-action.

Le désavantage des traces remplaçante est le crédit assigné à un état ne tient compte que de la visite la plus récente. Un grand nombre de visites répétés n'aura aucun effet supplémentaire. En revanche, on obtient un système plus stable.

3.2.3 Représentation de fonctions

Les algorithmes de programmation dynamique [24] forment une famille d'algorithmes qui ont été développés pour s'adresser aux problèmes où l'on doit évaluer différentes options en fonction de l'état courant. Ces algorithmes utilisent dans ce genre de situation une table disposant d'une entrée pour chaque paire de valeurs possibles de variable d'état et d'action choisie. La table est mise à jour chaque fois que l'action est choisie pour cette valeur précise de la variable d'état. Ceci devient rapidement impossible quand les dimensionalités des variables d'état et d'action à choisir deviennent un tant soit peu importantes [66][17] ou quand le problème comprend des variables ayant des valeurs continues [11]. De plus, une telle représentation ne permet aucune généralisation de l'expérience entre deux états semblables sans être identiques.

Pour toutes ces raisons, il est préférable d'utiliser un ensemble de paramètres libres pour faire l'approximation des fonctions, ou plus précisément, des réseaux de neurones artificiels dans notre cas [28][2][99]. Le réseau a comme entrées les valeurs des différentes variables d'état et produit à la sortie la valeur de la fonction V. Pour représenter la fonction Q, on peut soit avoir un réseau avec autant de sorties qu'il y a d'actions ou encore avoir autant de réseaux différents qu'il y a d'actions.

Le réseau apprend en modifiant ses paramètres dans la direction de l'erreur commise. Après qu'une action ait été choisie et exécutée, on obtient le signal de renforcement et soit l'évaluation de l'état suivant soit l'évaluation de l'option suivante choisie. Avec ces éléments en main, on peut déterminer l'erreur commise par le réseau. La dérivée de cette erreur est calculée par rapport à chaque paramètre du réseau. On peut finalement modifier ces paramètres dans la direction de leurs dérivées respectives.

3.3 Revue bibliographique

C'est au cours des années 80 que sont apparues les techniques modernes d'apprentissage par renforcement [105],[91]. Cependant, les travaux qui ont permis cet aboutissement remontent à beaucoup plus loin. Ils proviennent principalement de deux champs d'activités distincts. D'une part, les travaux actuels dans ce domaine ont été largement influencés par les avancements en psychologie, en particulier en psychologie animale. Ceux-ci ont permis les tout premiers pas en intelligence artificielle [102][69]. Par la suite, un grand nombre de travaux ont été publiés que l'on pourrait qualifier de techniques d'essais et erreurs. Nous reviendrons plus tard sur les réalisations qui ont marqué cette sphère d'activité. D'autre part, on trouve certains travaux à l'origine de l'apprentissage par renforcement dans la recherche portant sur le contrôle optimal et la programmation dynamique.

Les premiers travaux portant sur le contrôle optimal remontent à la fin des années 50. Bellman [24] est un des fondateurs de la programmation dynamique. C'est lui qui a appliqué ce principe au MDP (*Markov Decision Process*) [25]. En 1960, Howard [48] a introduit les méthodes itératives sur les politiques. Depuis, les techniques de programmation dynamique ont été étendues aux POMDP(*Partially Observable Markov Decision Process*). Lovejoy propose une revue des travaux concernant la programmation dynamique appliquée aux POMDP [61]. Bryson [52] fournit un historique des travaux portant sur le contrôle optimal.

Comme nous l'avons déjà mentionné, les techniques d'apprentissage par renforcement sont grandement inspirées à la base par les travaux en psychologie et en particuler en psychologie animale. En 1911, Thorndike [98] en arrivait à la conclusion que dans une situation donnée, les comportements d'un animal suivis d'une satisfaction, ou renforcement positif, avaient de meilleures chances de se reproduire dans une situation similaire. À l'inverse, devant une situation donnée, les comportements d'un animal suivis d'un inconfort avaient moins de chances d'être observés de nouveau. Plus tard, les travaux de Pavlov [75] sont venus confirmer la capacité des animaux, du moins les plus intelligents d'entre eux, d'apprendre à faire des associations entre des situations et des stimuli. Bien que ces principes aient été remis en question [53], ils ont inspiré les pionniers de l'intelligence artificielle tels Alan Turing [102] et Marvin Minsky [69].

C'est au cours des années 60 qu'est apparu le terme *apprentissage par renforcement* dans la littérature du domaine de l'ingénierie, notamment dans les publications de Waltz et Fu [104], de Mendel [65] et de Minsky [70]. Cependant, à cette époque, la distinction entre l'apprentissage par renforcement et l'apprentissage supervisé était plutôt floue. C'est notamment le

cas de Rosenblatt [78]. Bien que ce dernier se soit inspiré de toute évidence de l'apprentissage par renforcement - il employait des termes comme *récompense* et *punition* - les systèmes qu'il étudiait et développait étaient clairement des systèmes d'apprentissage supervisé. Lors de la phase d'entraînement, les données étaient accompagnées d'une indication sur la classe dont elles provenaient. La distinction entre ces deux disciplines diviendra plus claire au début des années 80 grâce au travaux de Barto *et al.* [18][20].

Si certains travaux réalisés au cours des années 60 ont été quelque peu oubliés, on le doit en partie à la confusion qui entourait ces sujets de recherche à cette époque. C'est le cas notamment des travaux de John Andreae. Andreae avait développé un système nommé STELLA qui interagissait avec son environnement par essais et erreurs [4]. Il s'était même intéressé aux problèmes des états cachés [5]. Cependant, ses travaux n'étaient pas très connus et n'ont donc pas beaucoup influencé les travaux de recherche en apprentissage par renforcement qui ont suivi.

En 1961, Donald Michie [67] présente un système qui apprend à jouer au tic-tac-toe par essais et erreurs appelé MENACE. En 1968, il présente en collaboration avec Chambers un système d'apprentissage par renforcement baptisé BOXES [68]. Ils apprennent entre autres à ce système, par des simulations, la tâche de la pôle en équilibre. Cette tâche consiste à contrôler les déplacements d'un chariot pouvant se déplacer selon une seule dimension. Sur ce chariot est attachée l'extrémité d'une tige pouvant faire des rotations autour du joint selon un seul degré de liberté. L'objectif de la tâche est de garder la tige en équilibre debout le plus longtemps possible. Cette tâche, reprise entre autres en exemple dans un article de 1983 de Barto *et al.* [19], a eu beaucoup d'influence par la suite sur les travaux de recherche en apprentissage par renforcement.

Au cours des années 70, Klopf a fait renaître l'intérêt pour la recherche sur les techniques d'apprentissage par renforcement avec ses travaux touchant les techniques par essais et erreurs [55][56]. Plusieurs chercheurs, dont John Holland [46], vont s'intéresser à plusieurs aspects de ces questions. Plus tard, Sutton [90] pousse plus loin les travaux amorcés par Klopf.

On peut associer avec l'apparition des algorithmes de différence temporelle (*Temporal Difference*) l'arrivée des techniques modernes d'apprentissage par renforcement. On attribue à Ian Witten le premier algorithme de cette famille. Bien qu'il n'utilise pas le terme *différence temporelle*, Witten présente, en 1977, un algorithme pour solutionner les problème d'environnement markovien à temps discret [109] appelé TD(0) tabulaire (*tabular TD(0)*). Au début des années 80, Barto, Sutton et Anderson proposent de remplacer les tables dynamiques, qui limitent beaucoup la taille des problèmes pouvant être pris en considération, par des approximateurs de

fonction, notamment des neurones artificiels, puis des réseaux de ces neurones [15][19][14].

C'est Sutton qui en 1988 introduit le terme de différence temporelle [91]. En 1989, Christopher Watkin présente l'algorithme Q-Apprentissage (*Q-Learning*) [105][106]. Cet algorithme utilisant la notion de différence temporelle est probablement l'algorithme d'apprentissage par renforcement le plus utilisé de nos jours.

Cependant, dès le début des années 70, les travaux de Klopf laissaient entrevoir l'arrivée éventuelle des algorithmes de différence temporelle. Schultz *et al.* retrace dans un article paru en 1997 les étapes qui ont mené à l'arrivée des algorithmes de différence temporelle [81].

Au cours des années 90, les techniques d'apprentissage par renforcement se sont raffinées. La variété des problèmes qui peuvent être abordée par ces méthodes ne cesse de s'accroître. Dès 1990, Ackley *et al.* [2] et Knoblock [57] publient des articles à propos de leurs réflexions sur les moyens qui pourraient être envisagés pour traiter des problèmes plus complexes que ceux étudiés jusqu'ici. La même année, Barto *et al.* publient un article [17] traitant de la complexité des problèmes pouvant être pris en considération par les techniques d'apprentissage par renforcement. Thurn *et al.* exposent certains risques quant à la convergence des algorithmes d'apprentissage par renforcement dans un article paru en 1993 [99]. Baird *et al.* proposent également en 1993 certains palliatifs pour solutionner le problème des espaces continus de hautes dimensions [11]. Littman *et al.* publient en 1995 un article qui vise à accroître la taille des problèmes qui peuvent être solutionnés, particulièrement les POMDP [66]. Le traitement des POMDPs au moyen des algorithmes d'apprentissage par renforcement fait également l'objet d'un article de Jaakkola *et al.* cette même année [101]. Toujours en 1995, Barto *et al.* s'intéressent à traiter des problèmes temps-réel [16]. Sutton *et al.* publient un important article en 1998 sur la façon dont peuvent être pris en considération les problèmes qui se déroulent selon des échelles temporelles variables [94]. C'est également en 1998 que Wiering *et al.* proposent une optimisation pour accélérer l'apprentissage de l'algorithme *Q-Learning* [64]. En 1999, Sutton *et al.* suggèrent une nouvelle approche qui consiste à apprendre par descente de gradient sur paramètres qui régissent l'ensemble de la politique [93].

Sutton et Barto ont publié en 1998 un livre qui porte sur l'apprentissage par renforcement sous tous ses aspects [92]. Ils explorent les différentes techniques, les ramifications dans les familles d'algorithmes et les équivalences mathématiques. Quelques exemples de cas concrets sont présentés en annexes. On y trouve de plus une bibliographie détaillée sur le sujet.

L'apprentissage par renforcement peut être appliqué à un grand nombre

de tâches. Le présent projet a pour but de solutionner la tâche décrite dans le prochain chapitre.

Chapitre 4

V-Ball

Ce chapitre trace les grandes lignes des règlements du jeu sur lequel nous nous penchons. *V-Ball* (pour *Virtual Ball*) est un produit commercial développé par Enzyme Digital Marketing, une entreprise basée à Montréal, Canada pour laquelle ce titre est la première expérience de production de jeu vidéo.

Comme il est de plus en plus répandu dans le domaine des jeux vidéo, ce jeu offre la possibilité de jouer avec d'autres usagers en réseau, soit par un réseau local soit par Internet. Cependant, il peut se produire qu'un joueur, faute de moyens techniques, ne puisse se joindre à une partie réseau. Il peut également se produire qu'un joueur veuille pratiquer seul avant d'aller affronter d'autres personnes. C'est là que nos agents entrent en jeu.

4.1 Survol

Il s'agit d'un jeu entièrement réalisé en trois dimensions se déroulant en temps réel. Les usagers ont le choix entre une perspective de première personne (vue de l'intérieur du personnage) ou de troisième personne (la caméra est située derrière le personnage). Le but principal de ce jeu est plutôt simple. De trois à six joueurs sont en compétition pour obtenir la possession d'un ballon dans un grand espace 3D. Ce terrain comprend des obstacles (roches, végétation, etc.) et de fortes dénivellations. Chaque joueur à la possibilité de marcher, courir et utiliser des habiletés spéciales. Les joueurs ne sont limités dans leurs mouvements que par une bordure qui entoure la zone légale de jeu. De plus, les joueurs sont soumis à certaines lois de la physique, soit la gravité et le frottement.

Au début de la partie, le ballon est remis à un joueur au hasard. Le

joueur ayant la possession du ballon marque des points pour chaque seconde au cours de laquelle il conserve le ballon. Il est en situation dite défensive. Le joueur en situation défensive est clairement identifié par une sphère flottant au-dessus de sa tête. Les autres joueurs cherchent à obtenir la possession du ballon. Pour ce faire, ils doivent soit entrer en contact avec le porteur du ballon ou abattre ce dernier. Ces joueurs sont en situation dite offensive.

La durée d'une partie est fixe, soit cinq minutes. Puisqu'un un nombre constant de points est distribué à chaque seconde, le nombre total de points octroyé au cours d'une partie est donc constant.

4.2 Les habiletés spéciales

Chaque joueur dispose au total de sept habiletés spéciales, soit trois qu'il peut utiliser lorsqu'il est en possession du ballon (habiletés défensives) et quatre qu'il peut utiliser lorsqu'un autre joueur est en possession du ballon(habiletés offensives). Il est impossible d'utiliser une habileté défensive en situation offensive et vice-versa. Ces habiletés sont choisies de la liste des habiletés spéciales avant le début de la partie et restent fixes pour la durée de la partie.

4.2.1 Les habiletés offensives

Chaque joueur choisit quatre des habiletés suivantes avant le début de la partie. Les habiletés offensives ont toutes le même effet, c'est-à-dire de causer des dommages à la victime. Seules diffèrent leurs apparences visuelles et certaines caractéristiques comme la portée et la couverture radiale. Certaines armes se dirigent droit devant le tireur alors que d'autres ont un effet tout autour de celui-ci.

- Boule de feu
- Jet d'acide
- Jet d'éclairs
- Tremblement de terre
- Pluie de météorites
- Essaim d'abeilles
- Vague de feu

4.2.2 Les habiletés défensives

Chaque joueur choisit trois des habiletés suivantes avant le début de la partie. Ces habiletés ont pour but de ralentir, d'aveugler ou de semer la confusion chez l'adversaire. Elles ont des caractéristiques qui sont propres à chacune.

- Mur
  Cette habileté fait apparaître un mur derrière le joueur qui l'utilise, ralentissant ainsi les poursuivants le suivant de près.

- Invisibilité
  Cette habileté rend son utilisateur invisible pendant quelques brefs instants.

- Mine éblouissante
  Cette habileté fait apparaître une mine derrière celui qui l'utilise. Si un autre joueur foule cette mine, il devient aveugle pendant une courte période de temps. Le joueur ayant jeté la mine peut lui-même la déclencher s'il rebrousse chemin et la foule par accident.

- Mine destructrice
  Cette habileté fait également apparaître une mine derrière celui qui l'utilise. Lorsqu'un joueur foule cette mine, elle explose et cause des dommages à tous les joueurs se trouvant dans un certain rayon. Le joueur qui a jeté la mine peut encore une fois lui-même la déclencher s'il marche dessus par mégarde ou s'il s'en trouve suffisamment près au moment où elle explose.

- Faux ballon
  Cette habileté a pour effet de faire apparaître un ballon au-dessus de la tête d'un autre joueur. Le but est évidemment de semer la confusion au sein des joueurs en situation offensive qui s'en prendront, s'ils ne s'aperçoivent pas de la supercherie, au mauvais joueur.

- Noirceur
  Cette habileté crée une zone de grande noirceur autour du joueur en situation défensive. Cette zone demeure pour toute son existence à l'endroit précis où elle a été créée, permettant entre-temps à son créateur de prendre la fuite.

- Brouillard
  Cette habileté fonctionne comme la Noirceur. Cependant, au lieu de la noirceur, c'est une zone de brouillard épais qui est créée.

- Dôme électrifiant
  Cette habileté génère autour du joueur un dôme d'électricité. Tout autre joueur qui se trouve à l'intérieur du dôme subit des dommages.

- Vitesse décuplée
  Cette habileté permet au joueur d'augmenter la vitesse à laquelle il se déplace pour une durée de temps limitée. Il peut donc ainsi semer ses assaillants plus aisément. Cependant, si la vitesse est plus grande, la navigation à travers les obstacles est plus difficile.

4.3 Le *Chi*

Chaque joueur possède une banque d'énergie qu'il peut utiliser pour déployer l'une ou l'autre de ses habiletés spéciales. On appelle cette énergie le *Chi* du personnage. Chaque joueur a la même quantité de *Chi* au début de la partie. Quand un joueur choisit d'utiliser une habileté spéciale, il doit déterminer la quantité de *Chi* qu'il veut y attribuer. Cette quantité de *Chi* sert ensuite à fixer un paramètre propre à l'habileté. Par exemple, dans le cas du jet d'acide et de la boule de feu, ce paramètre correspond au dommage causé au joueur qui reçoit le projectile. Pour le brouillard et la noirceur, ce paramètre correspond plutôt au temps que durera l'effet de l'habileté.

Une fois utilisé, le *Chi* est récupéré avec le temps. Le taux auquel le *Chi* est regagné est inversement proportionnel à la distance entre le joueur et le ballon. Le porteur du ballon récupère donc son *Chi* plus rapidement que les autres joueurs. Ceci compense un peu le fait que la plupart des attaques seront dirigées vers lui.

4.4 Les points de vie

Afin de mesurer le dommage accumulé par un joueur, un système de points de vie a été mis sur pied. Ces points de vie représentent de façon discrète l'énergie vitale dont dispose le joueur. Tous les joueurs commencent la partie avec le même nombre de points de vie. Chaque fois qu'un joueur est atteint par une arme, il perd une fraction de ses points de vie correspondant au *Chi* qui avait été affecté à l'utilisation de l'arme.

Quand un joueur n'a plus de points de vie, il est considéré abattu. Lorsqu'un joueur est abattu, il est ramené au milieu du terrain avec ses points de

vie et son niveau de *Chi* ramenés à leurs maximums possibles. Si le joueur en possession du ballon est abattu, le joueur ayant porté le dernier coup se voit octroyer la possession du ballon.

4.4.1 Le vol de points

En plus de perdre des points de vie, un joueur victime d'une attaque se fait également voler ses points de partie par son agresseur. Chaque fois qu'un joueur atteint d'un projectile un autre joueur n'ayant pas le ballon, une fraction des points du joueur atteint est transférée au tireur. La quantité de points transférée dépend encore une fois du *Chi* attribué au tir.

Cependant, si la victime ne possédait aucun point de partie, le tireur n'obtient aucun point. Si la victime possédait un nombre de points inférieurs à la quantité qui aurait dû être transférée, la victime perd l'ensemble des points qu'elle avait en banque au profit du tireur, qui lui ne touche que la part des points qui était disponible.

4.4.2 Régénération

À n'importe quel moment au cours de la partie, un joueur, qu'il soit en situation offensive ou défensive, peut choisir de régénérer ses points de vie. Ceci s'effectue en entrant dans une zone spéciale située au milieu du terrain. Une fois dans la zone, le joueur récupère ses points de vie selon un incrément constant à chaque seconde. Pour se régénérer, le joueur doit être seul dans la zone.

4.5 Les personnages

Le joueur peut choisir entre différents personnages. Il choisit un personnage avant le début de la partie et le conserve pour toute la durée de celle-ci.

Les personnages se différencient, outre leurs apparences, par la valeur de leurs caractéristiques spécifiques au jeu. Ces attributs sont la vitesse de marche, la vitesse de course, la vitesse de rotation, la vitesse de régénération des points de vie et des points de *Chi*. De plus, certains personnages ont des tirs plus précis que d'autres. Ces différences dans les caractéristiques sont équilibrées de façon à ce que tous les personnages soient de forces équivalentes.

Maintenant que les paramètres et les limites du problème sont définis, le

chapitre suivant présente l'architecture de la solution conçue dans le cadre du projet pour y répondre. Les détails concernant la planification de trajectoire et les algorithmes d'apprentissage sont présentés dans les deux chapitres subséquents.

Chapitre 5

L'architecture du système

Le présent chapitre précise l'architecture du système qui a été conçu. La planification de trajectoire est prise en charge par un module entièrement indépendant du module de prise de décisions. La description du module de planification de trajectoire fait l'objet du chapitre suivant. Le système dont l'architecture est traitée dans le chapitre actuel est dédié exclusivement à la prise de décision de haut niveau.

Les exemples utilisés dans ce chapitre se rapportent à l'architecture conçue pour traiter des parties à six joueurs. Cependant, une partie peut se jouer avec la participation de trois à six joueurs. Certains changements mineurs sont nécessaires pour traiter le cas des parties à trois, quatre et cinq joueurs, entre autres dans la taille de la variable d'état et dans le nombre d'unités des réseaux de neurones. Une seule architecture prenant en charge tous les cas était pratiquement impossible à concevoir. La stratégie optimale pour une partie à trois joueurs est différente de la stratégie optimale pour une partie à six joueurs. Le cas de six joueurs est retenu en exemple car il s'agit évidemment du cas le plus complexe.

5.1 Représentation de l'état et des actions

L'interface entre l'agent et le monde virtuel dans lequel il évolue comporte deux facettes principales. D'une part, la perception que l'agent a de l'univers est représentée par l'espace d'état. D'autre part, l'agent doit analyser l'information qui lui est disponible et prendre des décisions. Ces décisions s'expriment via l'espace des actions. Comme chaque agent est autonome et que la nature de la tâche étudiée fait qu'il n'y a aucune coopération directe entre les agents, l'espace d'état de chaque agent doit être complet en soit. De plus, les agents n'ont aucune information sur les décisions des autres agents

outre les impacts de celles-ci sur l'état du monde.

La tâche étudiée ici se caractérise par un nombre relativement grand de variables, qui sont pour la plupart continues, tant dans l'espace d'état que dans l'espace d'action. Un effort doit être fait afin de réduire le plus possible la taille de ces espaces.

5.1.1 Agents omniscients

Dans un premier temps, afin de simplifier le problème, il a été décidé que les agents étaient omniscients [42]. En effet, les agents ont accès en tout temps à toute l'information de la scène, notamment la position de tous les autres joueurs, même si ceux-ci se trouvent hors du champ de vision de l'agent. Ceci est vrai à l'exception du cas où un des joueurs devient invisible ou si l'agent est aveuglé. Dans ces cas-là, la position du joueur invisible ou la position de l'ensemble des joueurs si l'agent devient aveugle ne sont plus actualisées pour la durée de l'effet.

Cependant, on peut ajouter ensuite du bruit à l'information disponible aux agents. Cela a pour effet de rendre les décisions des agents plus crédibles aux yeux de l'usager. En effet, un agent qui prend en chasse un joueur situé à l'autre extrémité du terrain aura une idée approximative de la position du joueur, sans la connaître de façon précise. De plus, l'intensité du bruit peut être réglée pour calibrer les différents niveaux de difficultés offerts à l'usager.

Il aurait été possible de limiter l'information fournie aux agents à celle disponible dans son champ de vision. L'agent aurait eu ainsi droit à la même quantité d'information que les joueurs humains. Le problème aurait dû être formulé en terme de POMDP (*Partially Observed Markov Decision Process*). Il existe une vaste littérature touchant ce sujet [61][87], certains ouvrages proposant des solutions utilisant des algorithmes d'apprentissage par renforcement [101].

Un système permettant à l'agent d'inférer la position des autres joueurs aurait également été envisageable. À partir des observations faites sur la trajectoire d'un joueur ainsi que certaines autres informations pertinentes sur l'état de celui-ci, il aurait été possible pour l'agent de tenter de "deviner" la suite de la trajectoire empruntée par ce joueur. Évidemment, moins les observations sont récentes, plus la prédiction sur la position de l'adversaire risque de différer de la réalité. Un tel système aurait été fort complexe à mettre sur pied et probablement peu fiable. C'est pourquoi il a été décidé de permettre aux agents de "tricher" en leur donnant accès à certaines informations hors de leurs champs de vue.

Il est très fréquent dans l'industrie du jeu vidéo de permettre ainsi aux agents de "tricher" [110][111]. D'une part, cette pratique simplifie de beaucoup le travail de développement, évitant entre autres d'avoir à créer un module pour inférer la position des autres joueurs ou toute autre information non disponible, mais grandement utile, et de gérer un POMDP. D'autre part, bien que l'on cherche à créer une rivalité entre l'intelligence humaine et les capacités intellectuelles des agents, aucune comparaison n'est possible entre les deux. La force des agents réside dans leur capacité à traiter beaucoup d'information rapidement. Il est donc souhaitable de leur fournir cette information de façon à offrir un défi qui soit intéressant pour l'usager. Il est plus simple d'agir de façon stupide avec beaucoup d'informations que de façon brillante avec peu.

5.1.2 L'espace des états

Suite à une analyse de l'information qui était disponible à l'agent (positions des joueurs, directions des joueurs, possession du ballon, etc.), nous en avons dégagé les caractéristiques qui résument le mieux la scène pour prendre une décision par rapport à celle-ci.

L'espace d'état retenu est composé d'une variable booléenne, deux variables discrètes et vingt-quatre variables continues.

L'espace des états se défini par les variables suivantes :

- Possession du ballon (booléenne)
- Distance du ballon (continue)
- Distance du milieu (continue)
- Distance entre le milieu et le porteur du ballon(continue)
- Nombre de points de vie (discrète)
- Quantité d'énergie Chi disponible (continue)
- Nombre de points de parties accumulés (discrète)

De plus, pour chacun des cinq adversaires, nous avons les variables suivantes

- Distance entre l'agent et l'adversaire (continue)
- Distance entre l'adversaire et le porteur du ballon (continue)
- Distance entre l'adversaire et le milieu (continue)
- Produit scalaire entre la direction actuelle de l'agent et la direction vers l'adversaire (continue)

On peut remarquer la présence de certaines redondances, notamment pour l'agent qui est en possession du ballon. Nous verrons plus loin comment ce problème est contourné.

5.1.3 L'espace des actions

L'espace d'action est composé de sept variables booléennes, d'une variable discrète et d'une variable continue.

- Vitesse souhaitée (discrète)
- Direction souhaitée (continue)

De plus, pour chacune des quatre habiletés offensives et des trois habiletés défensives (voir chapitre 4), nous avons

- Activation de l'habileté spéciale (booléenne)

La vitesse des agents est discrétisée à trois niveaux : immobile, marche ou course. Le choix de la vitesse s'effectue dans le module de planification de trajectoire. Nous verrons les détails de cette décision plus loin.

Pour activer une habileté spéciale, l'agent doit d'abord donner la valeur **VRAI** à sa variable d'activation. À chaque incrément de temps durant lequel la variable a la valeur **VRAI**, une quantité du Chi de l'agent est retirée de la banque du joueur et est transmise à l'habileté. Ensuite, pour déclencher l'habileté, l'agent doit redonner la valeur **FAUX** à sa variable d'activation. Le Chi transféré entre temps servira à paramétriser l'effet l'habileté spéciale.

5.2 Hiérarchisation des décisions

Afin de réduire la complexité de la tâche selon l'approche "diviser pour régner", l'espace d'action disponible à l'agent a été discrétisé de manière hiérarchique [57][94][6]. Des actions de haut niveau ont été identifiées pour les situations offensives et défensives, donnant un ensemble de six options au porteur du ballon et neuf aux autres joueurs. Certaines des ces actions de haut niveau mènent directement à des actions spécifiques, d'autres sont subdivisées en un autre ensemble d'actions discrètes.

Les options retenues ici sont celles qui semblaient pouvoir présenter un certain intérêt à un point ou un autre de la partie et couvrir l'ensemble des situations pouvant se présenter à l'agent. Le but de l'exercice était uniquement d'identifier les options souhaitables sans pour autant déterminer de façon précise les conditions dans lesquelles chacune de ces options devrait ou ne devrait pas être choisie. Ce sont précisément ces conditions d'utilisation que l'ont veut faire apprendre aux agents. Dans certains cas, l'idée générale d'une situation où une option est souhaitable suit uniquement pour démontrer la pertinence d'une action.

Certaines options peuvent se dérouler sur plusieurs pas de temps. Par

exemple, si un agent veut tirer sur un joueur, il devra d'abord le viser avant de faire feu. Les décisions étant prises selon une fréquence constante, l'agent peut être appelé à prendre plusieurs décisions entre le moment où il décide de tenter de tirer sur un autre joueur et le moment où il tient ce joueur en mire. C'est pourquoi toutes les options peuvent, quand elles sont choisies, bloquer momentanément le processus de prise de décision jusqu'à ce qu'une condition de sortie, propre à chacune, soit remplie [6].

5.2.1 Actions du niveau supérieur

Les actions du niveau supérieur retenues pour la situation défensive et pour la situation offensive sont présentées dans le paragraphe qui suit.

Actions défensives du niveau supérieur

En situation défensive, les options du niveau supérieur sont les suivantes :

- Rester sur place
  Le joueur reste immobile. Cette option pourra être utile lorsque le joueur est isolé des autres joueurs et qu'il est hors de leurs champs de vue.

- S'enfuir
  On calcule la position moyenne des autres joueurs, pondérée par l'inverse de la distance avec l'agent. Le joueur fuit alors cette position moyenne.

- Régénérer
  Le joueur se rend au milieu du terrain de façon à récupérer ses points de vie, ou du moins une fraction de ceux-ci.

De plus, pour chacune des habiletés défensives, il existe l'option

- Utiliser l'habileté spéciale
  Cette option déclenche le processus d'accumulation de *Chi* et relâche l'habileté quand la quantité de *Chi* accumulée est jugée suffisante. On doit continuer de gérer les déplacements de l'agent jusqu'à ce que le niveau de *Chi* désirable soit atteint.
  Dans un premier temps, le choix de la quantité à affecter à l'utilisation d'une habileté spéciale se fait au moyen d'une fonction qui dépend de la quantité *Chi* disponible à l'agent, comprenant une composante qui est fonction du hasard. Cependant, la fonction régissant le choix de la

quantité de *Chi* pourrait éventuellement être apprise pour optimiser le gain obtenu à utiliser l'habileté spéciale.

Actions offensives du niveau supérieur

En situation offensive, les options du niveau supérieur sont les suivantes :

- Poursuivre le ballon
  Le joueur cherche à atteindre la position actuelle du porteur du ballon.

- Utiliser une habileté offensive contre le porteur du ballon
  Cette option détermine l'arme la mieux adaptée pour atteindre le joueur en possession du ballon, le met en joue, puis laisse partir le projectile quand le moment est venu.

- S'approcher du ballon
  Le joueur cherche à réduire la distance qui le sépare du porteur du ballon. Le but de cette manoeuvre est d'augmenter le taux avec lequel l'agent récupère son *Chi*. Bien que concrètement les actions qui découlent de cette option ressemblent beaucoup à celles de l'option *Poursuivre le ballon*, les conditions dans lesquelles cette option est souhaitable sont différentes. Ainsi, le soin de faire une distinction est laissé à l'agent.

- S'enfuir
  Il peut se produire des situations où un joueur, même s'il n'est pas en possession du ballon, a intérêt à s'enfuir ; c'est le cas, par exemple, s'il a accumulé beaucoup de points. Cette option fonctionne de la même façon que l'option du même nom élaborée pour la situation défensive.

- Régénérer
  Cette option correspond également aux mêmes actions que l'action défensive du même nom. Cependant, les conditions dans lesquels son exécution est jugée souhaitable pourront varier.

- Chercher le trouble
  Cette option fonctionne à l'inverse de l'option *S'enfuir*. On calcule la position moyenne des autres joueurs, pondérée par l'inverse de la distance avec l'agent. L'agent cherchera alors à atteindre cette position. Cette option peut être utile quand un joueur se retrouve à l'écart des autres joueurs, quand il apparaît au milieu de l'univers après avoir été abattu, par exemple(voir chapitre 4).

De plus, pour chacun des quatre autres joueurs qui ne sont pas en possession du ballon, il existe l'option

- Utiliser une habileté offensive contre un autre joueur
  Cette option fonctionne exactement comme l'option permettant de tirer sur le porteur du ballon.

Cinq options résument tous les contextes d'utilisation des habiletés offensives, soit une pour chacun des adversaires. Le choix de l'habileté spécifique à utiliser dans une situation se fait selon une fonction déterministe. Encore une fois, la fonction effectuant ce choix pourrait éventuellement faire l'objet d'un apprentissage.

On peut ainsi synthétiser l'utilisation des habiletés offensives en une seule option car les contextes d'utilisation de ces habiletés se rapprochent beaucoup les uns des autres, ce qui n'est pas le cas des contextes d'utilisation des habiletés défensives. De plus, les habiletés offensives visent un adversaire en particulier. Si on avait considéré chacune des quatre habiletés offensives individuellement pour chacun des cinq adversaires, on aurait obtenu vingt options. C'est donc pour maintenir au minimum la complexité du problème que l'on a choisit de regrouper ainsi l'utilisation des habiletés offensives.

5.3 Apprentissage

On cherche à faire apprendre aux agents à déterminer quelle est l'option la plus souhaitable pour une situation donnée. Pour ce faire, les agents doivent apprendre, pour chaque option, le contexte d'utilisation qui lui correspond. Ainsi les agents seront en mesure de déterminer, pour chaque situation, à quel contexte général cette situation se rapproche le plus.

Pour chacune de ces options, l'agent doit donc apprendre la fonction Q correspondante. Puisque la fonction Q représente en quelque sorte l'évaluation de la désirabilité d'une option face à une situation, ceci s'effectue, individuellement pour chaque option. Dans cette optique, des réseaux de neurones sont utilisés pour dresser les approximations des fonctions Q. Deux alternatives sont possibles pour accomplir cette tâche.

D'une part, chaque option peut disposer de son propre réseau de neurones entièrement dédié. Compte tenu que le but recherché est de maintenir la complexité à un minimum, l'espace d'état considéré peut être davantage réduit. Chaque réseau ne reçoit en entrée qu'un sous-ensemble des variables d'état regroupant les variables jugées pertinentes à l'évaluation de l'option correspondant à la fonction Q que le réseau apprend. On élimine toute redondance dans l'information fournie au réseau et cette information est présentée

de la façon la plus pertinente possible pour la décision qui doit être prise. Par exemple, la fonction Q avec le plus grand espace d'état dans notre système comprend six variables continues, deux variables discrètes et une variable booléenne. Cette approche simplifie beaucoup le traitement des habiletés défensives. L'utilisation de ces habiletés est apprise indépendamment pour chacune, puisque les contextes d'utilisation idéaux diffèrent largement de l'une à l'autre. En isolant chacune des options, il est simple de varier la nature des habiletés dont dispose l'agent, d'en remplacer une par une autre, d'en insérer une nouvelle, etc.

D'autre part, il est possible d'envisager l'utilisation d'un seul réseau qui dispose d'autant d'entrées qu'il y a de variables d'états et qui a autant de sorties qu'il y a d'options à évaluer. Cette alternative offre l'avantage potentiel de partager les poids à l'entrée. Ce qui est appris par l'expérience du choix d'une option peut se généraliser pour une autre option. Cependant, le gain obtenu par cette méthode n'est pas assuré et des vérifications empiriques doivent être effectuées. Par contre, cela complique le traitement des habiletés défensives. On peut penser qu'il faudra entraîner un réseau pour chaque sous-ensemble possible d'habiletés défensives. Il faudra imaginer une solution à ce problème si l'alternative d'utiliser un seul réseau de neurones s'avère nettement supérieure à la précédente.

Apprendre les fonctions Q simplifie grandement le problème. Une autre possibilité est d'apprendre les fonctions V et les probabilités de transitions pour chaque option. Mais ceci est beaucoup plus difficile, d'une part parce que la fonction V est très complexe, très stochastique et donc difficile à généraliser et d'autre part parce que les transitions entre les états sont également très stochastiques et sujettes à de larges fluctuations dues, en partie, à l'interaction avec un être humain.

Il existe une autre possibilité, un peu différente des deux précédentes. Au lieu d'astreindre les agents au seul apprentissage des fonctions Q, c'est toute la politique que l'on peut faire apprendre aux agents. Nous verrons plus loin les détails de cette avenue possible.

5.3.1 Poids initiaux

Afin d'accélérer le processus d'apprentissage, on donne aux poids des connexions entre les différentes unités des réseaux de neurones des valeurs initiales déterminées à l'avance. Les poids initiaux nous permettent d'incorporer des conseils aux agents [62]. Sages de ces précieux conseils, les agents pourront apprendre plus rapidement puisqu'on pourra pondérer l'importance relative qui devrait être accordée à chaque variable d'état. On pourra également donner une première appréciation des valeurs situées aux

extrémités de la plage que peut occuper une variable d'état. Par exemple, si la distance qui sépare l'agent du porteur du ballon est très grande, la poursuite n'est peut-être pas l'option la plus prometteuse.

Bien que ces poids initiaux puissent accélérer l'apprentissage, ils l'orientent également. Ils peuvent avoir l'effet pervers d'empêcher l'exploration dans une direction qui aurait pu se révéler intéressante. Afin de minimiser ce risque, les valeurs absolues des poids initiaux doivent être relativement petites. Idéalement, l'amplitude de ces valeurs doit être plus importante que celle des valeurs données au hasard aux poids pour lesquels on ne souhaite pas donner de valeurs initiales particulières, mais moins importante que celle à laquelle on s'attend pour les valeurs finales.

5.3.2 Traces d'éligibilité

L'utilisation des traces d'éligibilité nous permet une grande flexibilité dans l'architecture des réseaux utilisée. Une trace est maintenue pour chacun des paramètres de chacun des réseaux. Ainsi, quand un paramètre a été impliqué dans une décision, la trace qui lui correspond est remise à jour en conséquence. On peut donc différencier à l'intérieur d'un unique réseau les paramètres qui ont mené à ce choix de ceux qui n'ont pas été en cause.

L'utilisation de ces traces d'éligibilité permet de prendre en compte le fait qu'une décision peut avoir des conséquences sur la valeur du signal de renforcement plusieurs pas de temps plus tard. La tâche apprise comporte donc des renforcements avec retard.

5.4 Politique du choix des options

La politique du choix des options est l'ensemble du processus par lequel une option est choisie entre toutes celles qui s'offrent à l'agent. Elle comprend aussi bien la façon dont sont évaluées les options que la façon dont est prise la décision finale une fois les options évaluées. Une politique doit avoir comme caractéristique, entre autres, de balancer l'exploration et l'exploitation [92].

La phase d'exploration consiste à faire des choix qui ne sont pas toujours les plus prometteurs, mais qui en revanche permettent d'acquérir de l'expérience par la visite de nouveaux endroits dans l'espace des états, ou encore par l'essai de nouvelles possibilités dans des endroits où la solution optimale n'a pas encore été trouvée. La phase d'exploitation se limite à choisir l'option ayant obtenu la meilleure évaluation, en se fiant sur le fait que l'expérience déjà acquise se généralise au cas actuel.

Dans plusieurs cas, on cherche donc à obtenir un équilibre entre ces deux phases. On veut commencer à valider l'apprentissage réalisé dès que des généralisations émergent. Et même une fois l'apprentissage bien avancé, il doit toujours y avoir un souci de continuer à faire un minimum d'exploration, question de poursuivre plus loin l'apprentissage. Dans le cas étudié ici, cette constante évolution présente plusieurs avantages. D'une part, elle permet aux agents de s'ajuster au profil du joueur. Après une phase d'entraînement préliminaire tenue sur les lieux de développement, les agents apprendront une fois chez l'usager des stratégies spécifiques au style de jeu de ce dernier. Chaque joueur ayant son style, non seulement les situations auxquelles feront face les agents varieront, mais les conséquences de leurs actions pouront également varier puisque les réactions du joueur seront différentes. Ensuite, certains joueurs pourront trouver leur plaisir dans cet aspect d'entraînement d'agents.

Il existe dans la littérature plusieurs politiques qui offrent des avantages intéressants. C'est notamment le cas des politiques ε-glouton (*ε-greedy*) et *Softmax* décrites plus bas. Il existe aussi des politiques plus sophistiquées qui font appel à plusieurs modes d'évaluation des options dont les résultats peuvent être combinés de plusieurs façons [63]. Il est question à ce stade de comités d'experts où chaque mode d'évaluation est considéré comme un expert qui vote sur l'option qui devrait être sélectionnée. L'importance relative de chacun des votes par rapport aux autres peut être déterminée selon plusieurs modèles : parfaite démocratie, importance du vote selon la qualité des résultats obtenus en suivant cet expert, etc.

Plus récemment, un article portant sur un algorithme permettant l'apprentissage de la politique a été publié [93]. Nous explorerons également cette possibilité.

5.4.1 La politique ε-glouton

Une des politiques souvent utilisées pour des problèmes similaires à celui étudié ici est la politique ε-glouton [92]. Cette politique choisit l'option ayant obtenue la meilleure évaluation dans la plupart des cas. Cependant, les autres options sont choisies en fonction du hasard selon un facteur ε.

La valeur du facteur ε est plus grande au début de la phase d'entraînement, mais elle diminue graduellement tout au long de l'apprentissage pour en arriver presque à 0 quand l'entraînement est considéré comme étant complet. Ainsi, l'exploration est plus importante au début de la phase d'entraînement mais par la suite, ce souci d'exploration est graduellement diminué au profit de l'exploitation de l'apprentissage réalisé.

5.4.2 La politique *Softmax*

Cependant, nous avons plutôt opté pour la fonction *Softmax* [92]. Cette fonction attribue à chacune des options une probabilité non nulle d'être choisie qui est proportionnelle à son évaluation. On obtient ainsi une meilleure exploration des possibilités. De plus, cette exploration est faite en fonction de l'expérience déjà acquise.

La probabilité de choisir une option est calculée selon l'équation suivante.

$$P(A = a|s,t) = \frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}} \tag{5.1}$$

où n est le nombre d'options disponibles.

Le facteur τ, parfois nommé la température par analogie à l'entropie, régit la fréquence avec laquelle les options sous-optimales sont choisies. Plus ce facteur est grand, plus l'entropie est grande, plus l'agent explore les options sous-optimales. Encore une fois, ce facteur diminue tout au long de la phase d'entraînement. Évidemment, il ne doit jamais atteindre la valeur zéro car la fonction utilise ce facteur pour effectuer des divisions. Cette fonction nous assure un minimum d'exploration en tout temps.

5.4.3 Apprentissage de la politique

Sutton *et al.* proposent dans un article récent un algorithme qui permet d'apprendre l'ensemble de la politique [93] et non seulement les fonctions d'évaluation des options. Cet algorithme se veut une solution à plusieurs problèmes rencontrés par le passé avec les politiques plus classiques. L'approche dominante dans la littérature au cours de la dernière décennie a été d'utiliser des approximateurs de fonctions pour apprendre la valeur individuelle des options puis d'utiliser ces évaluations à l'intérieur d'une politique comme la politique ε-glouton, par exemple.

Bien que cette approche ait connu du succès avec certains types de problèmes, elle demeure limitée. D'abord, elle suppose que la politique recherchée est fondamentalement déterministe puisqu'une fois la phase d'exploration terminée, on choisira plus souvent qu'autrement l'option la mieux évaluée pour une situation donnée. Cependant, pour une grand nombre de problèmes, la politique optimale est fondamentalement stochastique. Ce que l'on cherche n'est pas l'identité de l'option optimale pour une situation, mais les probabilités qu'ont chacune des options d'être l'option optimale pour cette situation.

Ensuite, dans l'approche classique, de petits changements dans l'évaluation d'une option peuvent déterminer si l'option sera choisie ou non. De telles discontinuités dans la politique peuvent grandement ralentir, voir même empêcher la convergence de l'algorithme. En effet, si de petits changements ont pour effet de faire complétement basculer le choix entre deux options devant une situation donnée, il se peut très bien que le système oscillera entre ces deux options dans cette situation sans jamais déterminer laquelle est la plus souhaitable. Ainsi, il a été prouvé que l'approche classique ne convergera pas dans un grand nombre de problèmes [99][10][26].

On pose donc θ, un ensemble de paramètres libres, par exemple les poids des connexions d'un réseau de neurones régissant la politique suivie. En fait, on cherche à approximer au moyen de ces paramètres une fonction qui détermine les probabilités avec lesquelles chacune des options devrait être sélectionnée dans une situation donnée. L'apprentissage s'effectue alors par descente de gradient sur les paramètres régissant la politique. L'idée maîtresse de cet algorithme peut s'exprimer par l'équation suivante :

$$\Delta\theta \approx \alpha \frac{\partial \delta}{\partial \theta} \tag{5.2}$$

où δ est l'erreur calculée et α est un facteur inférieur à 1 qui représente le pas d'apprentissage. On calcule donc pour chaque paramètre la dérivée partielle de l'erreur par rapport à ce paramètre et on modifie sa valeur proportionellement à cette dérivée.

Le module de prise de décision décrit dans ce chapitre ne peut à lui seul régir les déplacements de l'agent. Une fois que la décision de haut niveau est prise, l'agent doit déterminer la trajectoire à emprunter pour atteindre son objectif. Le chapitre suivant présente le module responsable de la planification de trajectoire.

Chapitre 6

Planification de trajectoire

La planification de trajectoire est un sujet de recherche très actif. Les chercheurs oeuvrant sur ce problème proviennent de disciplines variées : robotique, géométrie computationelle, théorie des graphes. Il s'agit d'un problème très complexe où les applications sont aussi nombreuses que les solutions proposées. Dans le contexte des jeux vidéo, la puissance de calcul disponible ne permet pas à ce jour l'implantation de solutions parfaites impliquant des recherches exhaustives. Souvent, il n'est pas nécessaire de trouver le chemin optimal. Il faut donc savoir faire des choix judicieux qui permettront d'obtenir des résultats d'une qualité satisfaisante dans un temps de calcul minimum.

Ce chapitre propose d'abord un survol de la littérature portant sur la planification de trajectoire. On y présente les algorithmes les plus couramment utilisés. Ensuite, les solutions retenues pour notre système sont exposées.

6.1 Revue bibliographique

La planification de trajectoire est un sujet sur lequel il se fait beaucoup de recherche, notamment dans le domaine de la robotique [30][31] [35][73][59]. Dans ce domaine d'application, un grand nombre de travaux suggèrent l'utilisation d'un système de potentiel où les obstacles constituent des répulseurs et la cible est modélisée par un attracteur. Si l'on affecte un potentiel négatif à l'attracteur et un potentiel positif aux différents répulseurs, la planification de trajectoire peut se résumer à suivre le minimum de potentiel [43]. On peut alors pousser l'analogie et comparer ce type d'analyse à l'analyse du déplacement d'un électron dans un champ de potentiel ou encore à des travaux de mécanique des fluides.

On retrouve également un bon nombre de travaux portant sur la planification de trajectoire dans la littérature spécialisée dans la géométrie computationelle [103][7]. Ces travaux poussent très loin l'étude géométrique de l'agent, des obstacles et de la trajectoire désirée. Les solutions proposées sont souvent complètes, mais par contre, elles sont gourmandes en ressources. Il arrive souvent dans le domaine des jeux vidéo que l'on approxime la forme de l'agent et des obstacles par une sphère ou même par un point. Ceci réduit ainsi beaucoup l'utilité d'une analyse géométrique poussée.

Certains travaux s'inspirent du comportement de certains animaux pour tenter d'obtenir de nouveaux algorithmes [38][12]. Dans [29], Borst *et al.* proposent un algorithme inspiré par les observations des comportements tenus par des mouches.

Glasius *et al.* proposent une façon de calculer un flux de potentiel au moyen de réseaux de neurones [43]. De nombreux chercheurs explorent les façons de combiner les algorithmes d'apprentissage à la planification de trajectoires, en particulier par l'utilisation d'algorithmes d'apprentissage par renforcement [8][54].

Cependant, un grand nombre des algorithmes de planification de trajectoire disponibles dans la littérature ne peuvent être utilisés dans le domaine qui nous intéresse [77]. En effet l'aspect temps réel rend impossible l'utilisation d'algorithme de recherche exhaustive. De plus les ressources disponibles sont déjà grandement sollicitées par d'autres tâches, notamment l'affichage. Par contre, la nature du domaine comporte ses avantages. La trajectoire obtenue à la sortie de l'algorithme n'est pas tenue d'être la solution optimale. Le comportement d'adversaires d'un jeu est loin de ce que l'on pourrait appeler un système critique. La trajectoire suivie par un agent peut donc dévier un peu de la trajectoire optimale sans compromettre pour autant le divertissement obtenu par l'utilisateur du système. C'est pour ces raisons que dans le domaine des jeux vidéo, les algorithmes les plus répandus sont les algorithmes tirés de la littérature consacrée à la théorie des graphes. L'utilisation d'algorithmes provenant des travaux portant sur la théorie des graphes implique évidemment une discrétisation du terrain [89] pour les jeux qui, comme *V-Ball*, se jouent dans un espace parfaitement continu. En plus de nécessiter relativement peu de puissance de calcul, certains de ces algorithmes ont l'avantage d'être en mesure de traiter la possibilité d'avoir des coûts variables entre les points. On peut ainsi prendre en considération la dénivellation, le risque impliqué ou tout autre facteur pouvant influencer le choix d'un chemin autre que la simple distance euclidienne. L'algorithme par excellence issu de cette spécialité pour notre domaine d'application est de loin A^* [89][110][111].

6.1.1 L'algorithme A^*

L'algorithme A^* [89] est semblable à l'algorithme de Djikstra. Cependant, pour l'utiliser, il faut établir une heuristique qui permet d'évaluer le coût entre un point donné et le point d'arrivée. On calcule ainsi pour chaque noeud sa valeur

$$f(n) = g(n) + h(n) \tag{6.1}$$

où

f(n) est la valeur assignée au noeud n ;
g(n) est le coût minimal actuel pour arriver au noeud n à partir du point de départ ;
h(n) est l'évaluation heuristique pour atteindre le point d'arrivée à partir du noeud n. Cette évaluation ne doit cependant jamais être plus grande que la valeur réelle. La distance euclidienne peut donc très bien faire l'affaire.

Cet algorithme est généralement utilisé sur des espaces en deux dimensions discrétisés de façon régulière. Cependant, on peut très bien envisager un graphe représentant la discrétisation d'un terrain en trois dimensions. Chaque noeud, plutôt que d'avoir jusqu'à huit voisins possibles, peut en compter jusqu'à vingt-six (huit au même niveau et neuf sur les niveaux inférieurs et supérieurs).

```
Soit Ouvert, la liste des noeuds ouverts
Soit Fermé, la liste des noeuds fermés
Soit s, le noeud de départ
s.g = 0
s.h = EstiméDeLaDistanceAuBut( s )
s.f = s.g + s.h
s.parent = null
Ajouter s à Ouvert
Tant que Ouvert n'est pas vide
    Récupérer d'Ouvert le noeud n, tel que n est le noeud avec la valeur f la
plus basse des noeuds ouverts
    Si n est le noeud du but
        Construire le chemin
        Retourner la valeur succès
    Pour chaque successeur n' de n
        nouveaug = n.g + coût(n,n')
        Si n' est dans Ouvert ou Fermé,
        Et n'.g ≤ nouveaug
            Passer au successeur suivant
        n'.parent = n
        n'.g = nouveaug
        n'.h = EstiméDeLaDistanceAuBut( n' )
        n'.f = n'.g + n'.h
        Si n' est dans Fermé
            Retirer de Fermé
        Si n' n'est pas dans Ouvert
            Ajouter n' dans Ouvert
    Ajouter n dans Fermé
Retourner la valeur échec
```

FIG. 6.1 – Algorithme A*

Il existe une version dynamique de cet algorithme que l'on nomme *D** [88]. La version dynamique est utile lorsque l'on veut recalculer la trajectoire une fois que des changements ont été apportés à la scène, par exemple si certains obstacles ont été déplacés. L'algorithme permet d'utiliser les calculs déjà faits pour déterminer la trajectoire initiale et ainsi simplifier le traitement à faire pour modifier cette trajectoire. Cependant, cet algorithme nécessite évidemment davantage d'espace mémoire. C'est pourquoi il n'a pas été retenu dans le cadre du présent projet.

6.2 Planification de trajectoire de notre système

Dans notre système, la planification de trajectoire se fait parfaitement indépendamment de la prise de décision. Un module isolé du reste du système se charge de la planification de trajectoire une fois le processus de prise de décision complété.

Une fois qu'une décision dite de haut niveau est prise, on soumet au module de planification de trajectoire la position actuelle de l'agent ainsi que la position de la cible souhaitée et le module retourne l'angle dans lequel l'agent doit se déplacer au pas suivant. Le calcul de cet angle s'effectue en deux étapes. La première trace grossièrement le chemin que devra suivre l'agent. La seconde traite les détails de ce chemin.

Le module de planification de trajectoire est également en charge de déterminer la vitesse avec laquelle doit se déplacer l'agent. La façon dont est prise cette décision sera décrite plus loin dans le présent chapitre.

6.2.1 Chemin complet

La première étape du travail de planification de trajectoire est de tracer un chemin grossier mais complet de la position actuelle de l'agent jusqu'à sa destination finale. Ce chemin se calcule au moyen de l'algorithme A^* [89]. L'utilisation de cet algorithme implique nécessairement une discrétisation de l'espace. La granularité de cette discrétisation a évidemment un impact direct sur le temps de calcul nécessaire pour établir une trajectoire. Dans le contexte du domaine d'application qui nous intéresse, la grande sollicitation des ressources de calcul ainsi que l'aspect temps-réel font en sorte qu'il est impossible de mettre sur pied une trajectoire complète et très détaillée. C'est pourquoi la granularité de la discrétisation demeure relativement grossière. Cette première étape vise donc à atteindre deux objectifs.

En premier lieu, l'utilisation d'un algorithme comme A^* permet de savoir si un chemin reliant le point de départ et le point d'arrivée existe. Si un tel chemin n'existe pas, l'agent peut alors agir en conséquence soit en cherchant à atteindre le point qui lui est accessible étant le plus près de l'objectif soit en choisissant un tout autre objectif. Ensuite, cette étape sert également à traiter les principales discontinuités du terrain : les passerelles, les tunnels, les précipices, etc. De plus, on peut éliminer les chemins passant par des dénivellations trop prononcées et qui sont donc infranchissables par l'agent. L'agent ne peut tout simplement pas franchir une pente ascendante abrupte et fait une douloureuse chute lorsqu'il s'aventure sur une pente descendante trop escarpée.

Une planification de trajectoire complète et très détaillée, du point de départ au point d'arrivée a été laissée de côté pour plusieurs raisons. La première, évidemment, est le temps de calcul nécessaire pour obtenir un chemin complet. Ensuite, l'environnement est très dynamique. Des obstacles peuvent apparaître et disparaître. Les autres joueurs sont en mouvement constant. Un chemin longuement planifié peut rapidement devenir inutile. Finalement, les joueurs du jeu étudié ici disposent en tout temps de plusieurs options. Un joueur peut très bien, en cour de route vers un endroit, prendre une nouvelle décision qui l'amènera vers une destination totalement différente. Un trajet complet, du point de départ au point d'arrivée, tout aussi optimal soit-il, devient alors désuet. Le calcul d'un chemin complet est donc un luxe qu'il est inutile de se payer pour notre cas.

Cette première étape ne considère que le terrain. Les obstacles se dressant sur la route ainsi que les autres agents pouvant représenter un danger ne sont traités que lors de l'étape suivante qui détermine les détails de la trajectoire.

6.2.2 Détails de la trajectoire

Afin de traiter des détails de la trajectoire, une fois que le chemin principal est tracé, le module de navigation possède sa propre représentation interne du monde dans laquelle chaque obstacle est modélisé par un ou plusieurs répulseurs. Ces répulseurs sont stockés dans différentes listes correspondant à différentes régions du terrain. Ainsi, l'algorithme ne tient en compte que les obstacles de la région où se trouve l'agent (ainsi que les régions mitoyennes lorsque l'agent se trouve proche des frontières). On gagne ainsi un temps de calcul précieux. Le module du vecteur de répulsion considéré pour chaque obstacle est inversement proportionnel à la distance entre l'obstacle et l'agent. Le module peut également obtenir la position des autres agents. Des répulseurs supplémentaires sont ajoutés lorsque ceux-ci sont suffisamment près de l'agent et qu'ils représentent un risque potentiel pour l'agent ou du moins un obstacle sur son chemin.

Une fois que tous les répulseurs et l'attracteur à considérer ont été placés, un calcul vectoriel est effectué afin d'obtenir l'action combinée des champs de potentiel de chacun de ces éléments. L'angle résultant de ce calcul est le vecteur de direction qui sera finalement retourné par le module. Une autre technique qui a été considérée consistait, une fois que l'ensemble des répulseurs et que l'attracteur avaient été placés, à échantillonner le potentiel d'une vingtaine de points situés sur la circonférence d'un cercle placé autour de l'agent ayant pour rayon la distance parcourue par l'agent en un pas. Le point dont le potentiel est le plus bas aurait été la direction la plus avantageuse pour l'agent. Cependant, cette technique était plus lourde en terme de calculs à effectuer et impliquait une discrétisation de l'ensemble

des directions que peut emprunter l'agent.

L'utilisation d'un tel algorithme présente l'inconvénient évident de ne permettre aucune aucune planification à long terme. L'agent suit pas à pas le chemin du potentiel minimum. Mais ceci importe peu puisque la planification à long terme est déjà complétée à cette étape. Cependant, cette technique ne requiert pas beaucoup de puissance de calcul. L'agent fait face à plusieurs dangers. Il peut tomber dans un minimum local, rester bloqué entre plusieurs minimums locaux, suivre un chemin non-optimal, etc. Cependant, la première étape de planification s'occupe en grande majorité de ces problèmes. En évitant d'inclure des concavités dans les obstacles et dans la morphologie du terrain, on peut également limiter les risques d'obtenir des minimums locaux.

6.2.3 Les dénivellations

La dénivellation n'est pas considérée dans le calcul du chemin, à moins qu'il s'agisse d'une pente trop abrupte impossible à franchir pour l'agent. Quand l'agent suit une pente vers le bas, il gagne de la vitesse. Quand il remonte une pente il perd de la vitesse. Dans l'implantation des lois de la physique du moteur de jeu auquel notre système se rattache, ces deux effets s'équilibrent. Donc, par le principe de conservation de l'énergie, pour aller d'un point à un autre, le chemin n'a aucune importance du point de vue des pertes ou des gains d'énergie causés par les pentes.

6.2.4 Choix de la vitesse

Comme nous l'avons déjà mentionné, le choix des vitesses auxquelles peut se déplacer l'agent est discrétisé à trois niveaux : immobile, marche ou course. Une fois qu'il a décidé qu'il devait se déplacer, qu'il a choisi la direction du déplacement, l'agent doit ensuite choisir la vitesse du déplacement à effectuer. Dans la plupart des cas, l'agent choisira la vitesse maximale puisqu'il atteindra ainsi plus rapidement son objectif, peu importe s'il cherche à atteindre une position spécifique ou simplement accroître ou réduire la distance le séparant d'un autre agent. Le jeu ne comporte aucun élément d'épuisement physique ou de limite de ressources énergétiques. Cependant, dans certains cas, il est préférable de ne pas se déplacer à la vitesse maximale.

Deux éléments sont pris en considération dans cette décision. D'abord, on calcule la différence angulaire entre la direction dans laquelle fait face l'agent et la direction du déplacement souhaité. Si cette différence est minime, aucune restriction ne s'applique à la vitesse. Cependant, si l'agent doit faire un demi-tour sur lui-même avant de partir dans la direction souhaitée,

il a avantage à ne pas se déplacer pendant la rotation. On fixe donc deux paliers dans la plage des différences angulaires possibles : un palier d'où il vaut mieux marcher que courir et un autre d'où il vaut mieux ne pas se déplacer que de marcher. Ces paliers sont différents pour chacun des personnages car ils dépendent des vitesses de marche, de course et de rotation.

6.2.5 Présence d'obstacles

Ensuite, la présence d'obstacles vient influencer le choix de la vitesse. Il n'est pas avantageux de voyager à une très grande vitesse dans un endroit où se trouvent de nombreux obstacles et donc où de fréquents changements de cap sont à prévoir. Ceci est encore plus vrai dans notre cas puisque les décisions de navigation ne sont pas prises instantanément, mais bien selon un intervalle régulier. Il faut donc un minimum de prévoyance pour éviter toute collision entre le moment d'une décision et la décision suivante. De plus, l'agent qui se déplace dans des zones criblées d'obstacles doit souvent dévier considérablement de la direction idéale pour éviter ces obstacles. Comme on cherche à minimiser les déplacements dans les autres directions et que, encore une fois, des décisions de navigation ne sont pas prises à tous les instants, il y a avantage à ralentir dans les zones à forte concentration d'obstacles. Pour juger de la concentration d'obstacles de l'endroit où il se trouve, l'agent observe le module du vecteur de répulsion calculé pour déterminer la direction à suivre, sans prendre en considération les vecteurs correspondants aux autres joueurs. L'agent évite ainsi de ralentir inutilement en présence des autres joueurs. Encore un fois des paliers, variables pour chacun des personnages, sont fixés.

Finalement, on combine l'effet de ces deux facteurs pour déterminer la vitesse de déplacement qui convient le mieux à la situation actuelle de l'agent.

Maintenant qu'un apperçu de système dans son ensemble et dans certains de ses détails a été donné, nous pouvons nous pencher sur la démarche expérimentale qui a été suivie par rapport aux algorithmes d'apprentissage.

Chapitre 7

Démarche expérimentale

Cette section retrace la démarche suivie. Elle présente l'ensemble des expériences réalisées au cours du projet visant à obtenir les agents les plus performants possibles pour la tâche qui nous intéresse, ainsi que la méthode d'entraînement la plus efficace pour faire apprendre à nos agents leur tâche.

À moins d'indications contraires, les expériences décrites dans ce chapitre ont été réalisées avec des parties à quatre joueurs. Il en a été décidé ainsi pour éviter de diluer les tendances. Si un agent s'avère légèrement supérieur aux autres, la présence de joueurs supplémentaires ajoute du bruit aux résultats. Il est beaucoup plus difficile pour un joueur de se démarquer de cinq adversaires que de trois. Les cas des parties à trois, cinq et six joueurs peuvent alors servir pour valider les résultats obtenus dans les parties à quatre joueurs.

De plus, afin d'éviter des distorsions dans les résultats causés des facteurs autres que ceux que nous cherchons à étudier, tous les joueurs de nos parties simulées joueront avec le même personnage. Ils auront ainsi les mêmes valeurs pour les attributs de vitesse de déplacement, de vitesse de rotation, de vitesse de régénération, etc. Encore une fois, l'utilisation de valeurs différentes pour ces attributs pourra servir dans une phase ultérieure de validation des conclusions tirées.

7.1 L'heuristique

La première étape est de mettre sur pied une heuristique pour le jeu sous étude. Il s'agit donc de réfléchir de façon à résoudre le problème au moyen d'algorithmes plus classiques. La politique de prise de décision est donc entièrement explicitée par les algorithmes conçus et aucun apprentissage ne peut alors avoir lieu. Cette démarche vise à rencontrer deux objectifs. Le

premier est simplement d'acquérir une meilleure compréhension du problème et ainsi d'obtenir des connaissances qui lui sont spécifiques. Notamment, cette heuristique nous sert au moment d'identifier les variables d'état les plus pertinentes et ainsi en faire des entrées pour les réseaux de neurones. C'est également par ce travail que nous sommes en mesure de choisir les valeurs initiales des poids des connexions entre les unités de ces réseaux.

Ensuite, les agents utilisant cette heuristique comme politique peuvent par la suite être utilisés pour mesurer les performances des agents utilisant les algorithmes d'apprentissage. Il suffit de comparer les performances des agents suivant d'autres algorithmes quand on les oppose aux agents agissant selon cette heuristique pour déterminer quels algorithmes et quelles architectures de réseaux de neurones fonctionnent le mieux.

Cette heuristique s'appuie sur la même architecture que les agents ayant la capacité d'apprentissage. Seuls les réseaux de neurones chargés d'évaluer les options en dressant une approximation la fonction Q doivent être remplacés par des algorithmes conventionnels. Il nous faut donc concevoir des algorithmes pour évaluer chacune des options qui s'offrent aux agents. Pour ce faire, chacune de ces options est analysée individuellement afin de déterminer pour chacunes les conditions dans lesquelles elles constituent un choix raisonnable. Nous paramétrisons ensuite les fonctions d'évaluation de façon à obtenir des évaluations à l'intérieur de certaines échelles.

En guise d'exemple, nous présentons ici l'algorithme qui a été conçu pour évaluer l'option *Rester sur place*. Cette option du joueur en situation défensive consiste simplement à demeurer immobile. Elle devrait être choisie quand une bonne distance sépare l'agent de son adversaire le plus près. La fonction d'évaluation de cette option développée pour l'heuristique est

$$Evaluation = \frac{(Dist\_min - Distance\_de\_confort)}{Diametre\_du\_terrain} \tag{7.1}$$

où Dist_min est la distance du plus proche adversaire et Distance_de_confort est la distance minimale pour qu'un adversaire ne représente plus une menace. Cette dernière distance est déterminée empiriquement en observant les agents jouer entre eux.

7.2 Plate-forme d'entraînement

Afin d'accélérer le processus d'apprentissage ainsi que les essais d'évaluation, il nous faut bâtir ce qu'on appelle une plate-forme d'entraînement. En fait, il s'agit d'une version du moteur du jeu de laquelle certains éléments ont été

retirés afin d'accélérer la vitesse d'exécution.

Parmi les éléments retirés, l'affichage est probablement celui qui donne le meilleur gain en performances. Bien que du matériel dédié prenne en charge une grande partie de cette tâche, l'affichage demeure un des éléments les plus gourmands en ressources, autant en temps de calcul qu'en espace mémoire nécessaire. Il est également possible de délaisser les éléments sonores. Ceci permet malgré tout un gain non négligeable dans les performances puisque les sons sont positionés en trois dimensions. Lorsqu'un son est entendu par l'usager, l'échantillon sonore est joué de façon à donner l'illusion qu'il provient réellement de la position d'où il est sensé être émis. Ce positionement dans l'espace nécessite quand même une certaine utilisation des ressources puisque le jeu comprend de nombreux événements sonores.

Cependant, certaines tâches du moteur du jeu n'ont pu être retirées. C'est notamment le cas des détections de collisions. La détection de collision est primordiale au fonctionnement du jeu, par exemple pour savoir si deux joueurs sont entrés en contact et donc que le ballon doit changer de mains. La détection de collision donne lieu à des calculs géométriques 3D relativement lourds, particulièrement quand on cherche à développer une application temps-réel capable de calculer entre 30 et 60 images par secondes.

Ceci explique pourquoi le gain final de performances pour la plate-forme d'entraînement par rapport au moteur original reste limité. Plutôt que de calculer le temps écoulé depuis la dernière image qui a été affichée et d'effectuer les déplacements dans la scène par rapport à ce temps, nous avons gardé l'incrément de temps entre deux images constant en terme de temps de jeu. Pour l'observateur, l'écoulement du temps semblerait non linéaire puisque certaines images sont plus longues à calculer que d'autres et que le pas dans le temps pris après chacune de ces images est constant. À une cadence de 25 pas de temps par seconde (temps de jeu), un minimum pour s'assurer d'obtenir des collisions fidèles notamment entre les projectiles et les joueurs, et sans tenir compte de ce qui ne sert qu'au divertissement de l'usager (sons et images), le temps s'écoulait entre 5 et 6 fois plus rapidement que le temps-réel.

7.3 Architecture de base

En premier lieu, une architecture de base est construite. Cette architecture semble, à première vue, être une hypothèse intéressante pour ce qui est de l'architecture optimale que nous cherchons à déterminer. Cette architecture de base utilise l'algorithme *Q-Learning*. L'évaluation de chaque option est réalisée grâce à son propre réseau de neurones indépendant des autres.

Tous ces réseaux comprennent une couche cachée. On ajoute un élément de non-linéarité en utilisant une tangente hyperbolique pour le calcul de la valeur de sortie. Le nombre d'unités cachées a été fixé au double du nombre des entrées pour commencer. Nous étudirons plus tard la possibilité d'utiliser d'autres quantités.

7.4 Échantillonnage des parties

Nous avons voulu nous assurer, avant de savoir si les agents sont en mesure de généraliser l'expérience acquise, qu'ils soient capables d'apprendre des exemples précis quand ceux-ci leur sont répétés à plusieurs reprises. Pour ce faire, il faut échantillonner la totalité des coups joués par l'ensemble des joueurs. On entend par un coup joué par un agent l'option de haut niveau choisie pour un tour. Bien que l'action du jeu se déroule en temps-réel, le processus de décision des agents est déclenché selon un intervalle régulier. De plus, une fois la décision de haut niveau prise, le reste du processus de prise de décision est ajusté de façon à être complètement déterministe. Nous avons donc donné aux agents la possibilité de sauvegarder, sur une base individuelle, la suite de leurs décisions ainsi que la possibilité de rejouer ces coups dans le même ordre. En bout de ligne, puisqu'il en est ainsi pour chacun des joueurs, les états qui sont présentés à l'agent sont exactement les mêmes et donc les valeurs fournies aux entrées des réseaux de neurones sont elles aussi les mêmes.

Pour observer l'apprentissage fait pour un nombre restreint de situations, nous faisons jouer aux agents cinq parties différentes. Par la suite, les agents rejouent chacune de ces 5 parties à 50 reprises. Pour mesurer la progression réalisée par les agents au cours de ces 50 reprises, nous avons utilisé l'erreur quadratique.

7.4.1 Mesure de l'erreur

Théoriquement, la valeur du choix d'une option (la valeur de la fonction Q) équivaut à la sommation de l'ensemble des signaux de renforcement r reçus par la suite, multiplié par un facteur d'atténuation γ qui diminue exponentiellement avec la distance temporelle.

$$Q_t(s_t, a_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{7.2}$$

Par le principe du *bootstrapping*, on peut utiliser l'évaluation de l'option

suivante choisie pour établir l'approximation de la sommation des valeurs futures du signal de renforcement. On obtient donc :

$$Q_t(s_t, a_t) = r_{t+1} + \gamma Q_{t+1}(s_{t+1}, a_{t+1}) \tag{7.3}$$

L'erreur peut donc être mesurée par :

$$Erreur = 0.5(Q_t(s_t, a_t) - r_{t+1} - \gamma Q_{t+1}(s_{t+1}, a_{t+1}))^2 \tag{7.4}$$

7.4.2 Résultats

Les figures 7.1 à 7.5 présentent les courbes de l'erreur commise par les quatre joueurs, tous en situation d'apprentissage, pour cinq parties échantillonnées. Chaque époque correspond à une partie de cinq minutes (temps de jeu). Comme en témoignent les graphiques, l'erreur diminue avec le temps, ce qui permet de croire que les agents apprennent effectivement les exemples qui leur sont présentés.

On note que la différence entre les erreurs commises par chacun des agents varie d'une partie à l'autre. Cette différence s'explique par le fait que les situations rencontrées par chacun des agents au cours d'une partie sont très différentes. Certaines situations sont plus complexes à analyser, donc à apprendre, que d'autres. Les agents qui ont fait face au cours de la partie apprise à des situations plus complexes commettront des erreurs plus importantes. Cependant les formes des courbes sont très similaires d'une partie à l'autre, d'un agent à l'autre. En général, la courbe d'apprentissage atteint une asymptote après une douzaine d'époques d'entraînement. Peut-être la capacité des réseaux est-elle insuffisante pour rduire davantage cette erreur.

7.5 Algorithmes d'apprentissage

Nous voulons établir l'algorithme d'apprentissage le mieux adapté à notre problème. À ce sujet, plusieurs possibilités s'offrent à nous. Nous explorons donc les plus prometteuses d'entre elles.

Tout d'abord, nous voulons comparer deux variantes de la famille d'algorithmes *TD(λ)* [92]. Il s'agit de *Sarsa* [85][79] et *Q-learning*[106]. La différence principale entre ces deux algorithmes réside dans la façon dont sont traités les choix sous-optimaux. Dans les deux cas, il arrive, par besoin d'explorations, que l'agent choisisse d'exécuter une option qui n'est pas la mieux

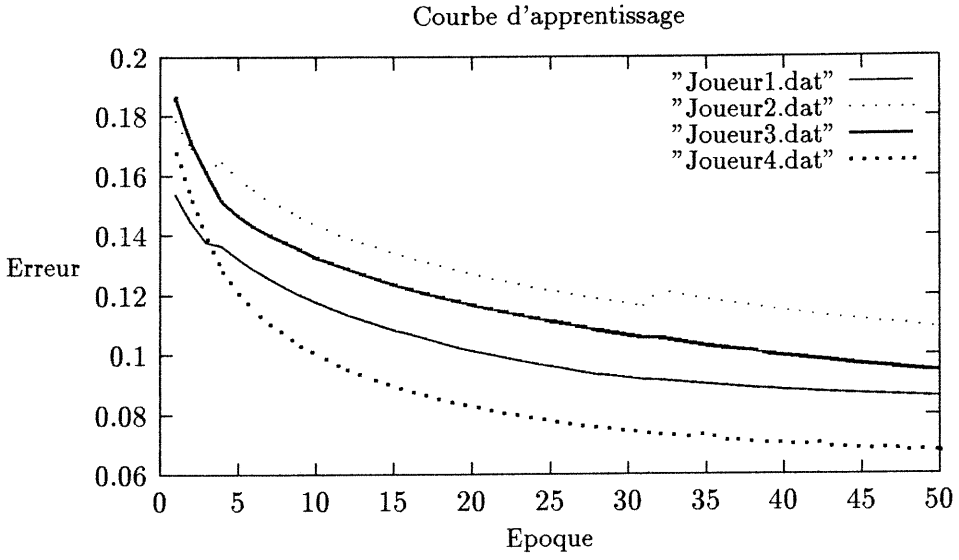


FIG. 7.1 – Partie échantillonnée 1

évaluée pour une situation donnée. Pour *Sarsa*, on traite ce choix comme n'importe quel autre. L'évaluation courante, bien que sous-optimale, est utilisée comme approximation des récompenses futures pour l'option qui avait été choisie au coup précédent. Ceci pourra avoir comme effet de diminuer, du moins temporairement, l'évaluation de l'option précédante. À l'opposé, dans *Q-learning*, aucun de ces choix n'est pris en ligne de compte dans l'apprentissage des fonctions Q. Aucune modification n'est apportée aux paramètres dans ces cas-là. Les deux approches ont leurs mérites. L'algorithme *Sarsa*, en prenant en considération les choix sous-optimaux, peut finir par produire des approximations qui sont plus proches des résultats réellement obtenus par la politique actuellement suivie. Cependant, puisqu'on diminue graduellement les chances que les options sous-optimales soient choisies, *Q-learning* fournit des approximations qui peuvent être plus près de la politique optimale qui devrait être suivie.

Ensuite, nous faisons également l'essai d'un algorithme récemment présenté par Sutton *et al.* [93] Cet algorithme a comme principe sous-jacent de faire de la descente de gradient sur l'ensemble de la politique suivie et non sur l'évaluation individuelle des différentes options qui s'offrent à l'agent. Il a été démontré que cet algorithme est assuré de converger vers une solution. Nous présentons plus loin les principales équations qui le composent.

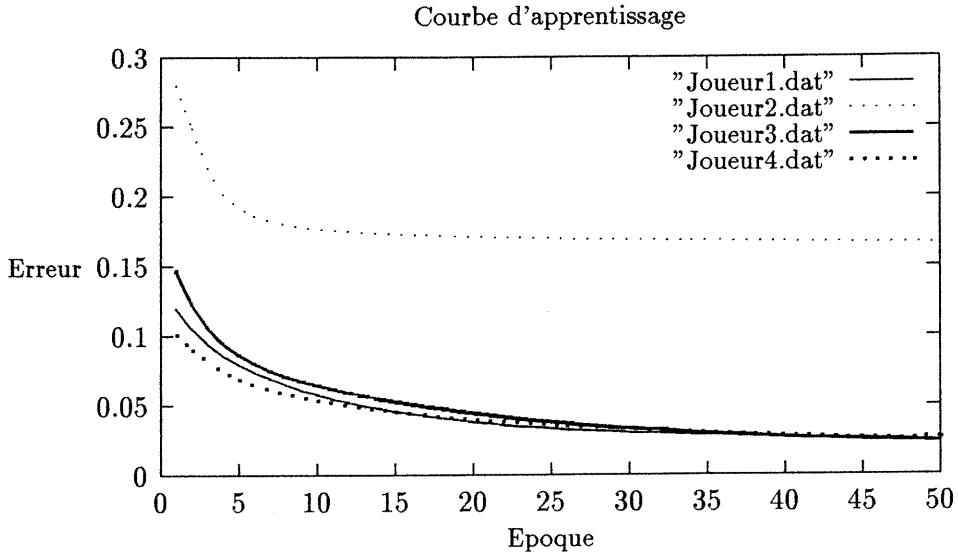


FIG. 7.2 – Partie échantillonnée 2

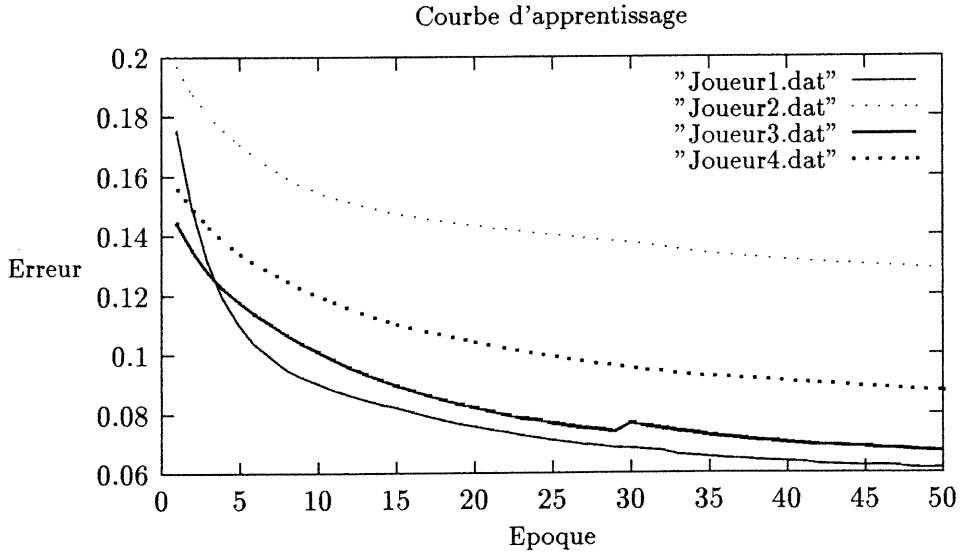


FIG. 7.3 – Partie échantillonnée 3

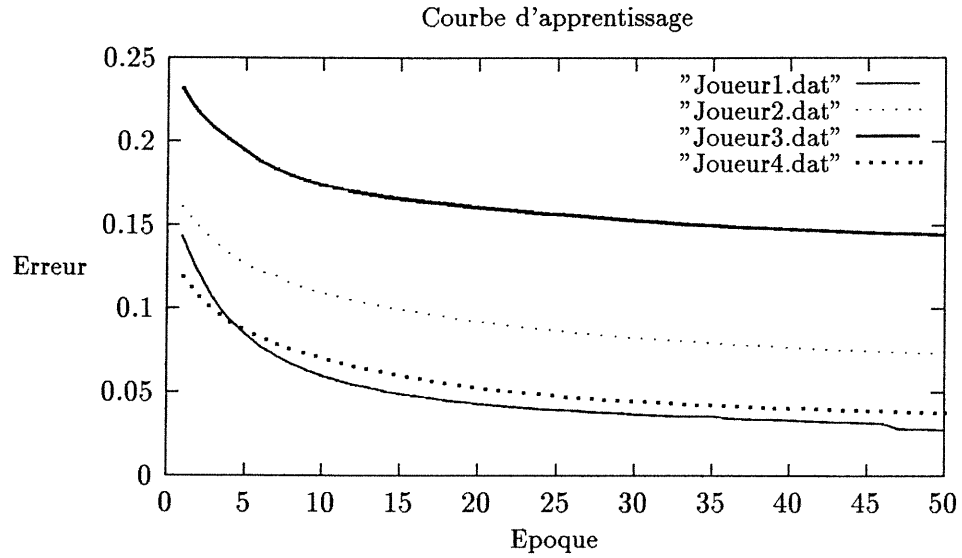


FIG. 7.4 – Partie échantillonnée 4

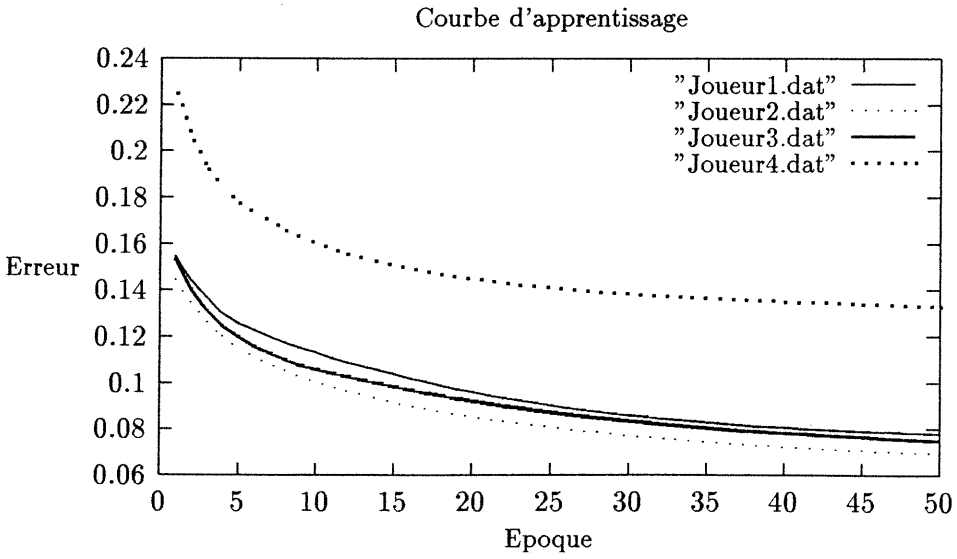


FIG. 7.5 – Partie échantillonnée 5

7.6 *Sarsa*

Nous présentons ici l'algorithme Sarsa par descente de gradient. Nous avons inclus à l'algorithme aussi bien les portions nécessaires à l'utilisation des traces remplaçantes que celles pour les traces accumulantes(voir section 3.2.2). Des tests préliminaires nous ont permis de déterminer rapidement que les traces accumulantes ne fonctionnent pas bien dans notre cas. Certaines options sont choisies beaucoup plus fréquemment que d'autres. Les valeurs des traces d'éligibilité correspondant à ces options grimpaient sans cesse. Les valeurs des poids des connexions augmentaient de façon démesurée. Ceci provoquait des valeurs de sortie en dehors des plages prévues. Ces valeurs sont utilisé dans l'algorithme pour le calcul de l'erreur. Aprés quelques itérations, les valeurs de sortie n'étaient même plus dans la plage de valeurs traitables par l'implantation de l'algorithme. En résumé, les réseaux de neurones s'emballaient et divergeaient, ce qui produisait finalement un arrêt brusque de l'exécution de la simulation.

Initialiser $\vec{\theta}$ arbitrairement et $\vec{e} = \vec{0}$
Répéter (pour chaque épisode)
  Soit s, l'état initial de l'épisode
  Soit a $\leftarrow argmax_a Q_t(a, s)$
  Selon une probabilité ε : a $\leftarrow$ action au hasard $\in A$(s)
  Répéter (pour chaque pas de l'épisode)
    $\vec{e} \leftarrow \gamma\lambda\vec{e}$
    $\forall \bar{a} \neq a$ (bloc optionel pour les traces remplaçantes)
      $\forall$ i $\in$ F$\bar{a}$
        e(i) $\leftarrow$ 0
    $\forall i \in F_a$
      e(i) $\leftarrow$ e(i) + 1 (traces accumulantes)
      OU e(i) $\leftarrow$ 1 (traces remplaçantes)
    Prendre l'action a, observer la récompense r et l'état suivant s'
    $\delta \leftarrow r - Q_t(a, s)$
    Soit a' $\leftarrow argmax_a Q_{t+1}(a, s')$
    Selon une probabilité ε : a' $\leftarrow$ action au hasard $\in A$(s')
    $\delta \leftarrow \delta + \gamma Q_{t+1}(a', s')$
    Calcul de $\Delta\vec{\theta}$. Pour chaque élément θ_i
      Calculer $\frac{\partial\delta}{\partial\theta_i}$
    $\theta_i \leftarrow \theta_i + \alpha\Delta\theta_i e_i$
    a $\leftarrow$ a'
  Jusqu'à ce que s' soit terminal

FIG. 7.6 – Algorithme Sarsa

où $\vec{\theta}$ est l'ensemble des paramètres à être appris, $\vec{e}$ est l'ensemble des

traces d'eligibilité de ces paramètres, γ est un facteur d'atténuation temporel, λ est le pas d'apprentissage, δ la mesure de l'erreur et F_a est l'ensemble des traces d'éligibilité relatives à l'action a.

Le calcul de $\Delta\vec{\theta}$ s'effectue par rétro-propagation du gradient à travers l'ensemble des unités composant le réseau.

7.6.1 Performances de l'algorithme Sarsa

Nous entraînons quatre agents suivant cet algorithme en les opposants pendant 100 époques. Encore une fois, l'erreur quadratique est une des mesures utilisées pour quantifier les performances. La moyenne des erreurs commises par l'ensemble des agents est calculée pour chaque époque d'entraînement. La figure suivante présente l'évolution de cette erreur au cours de l'apprentissage.

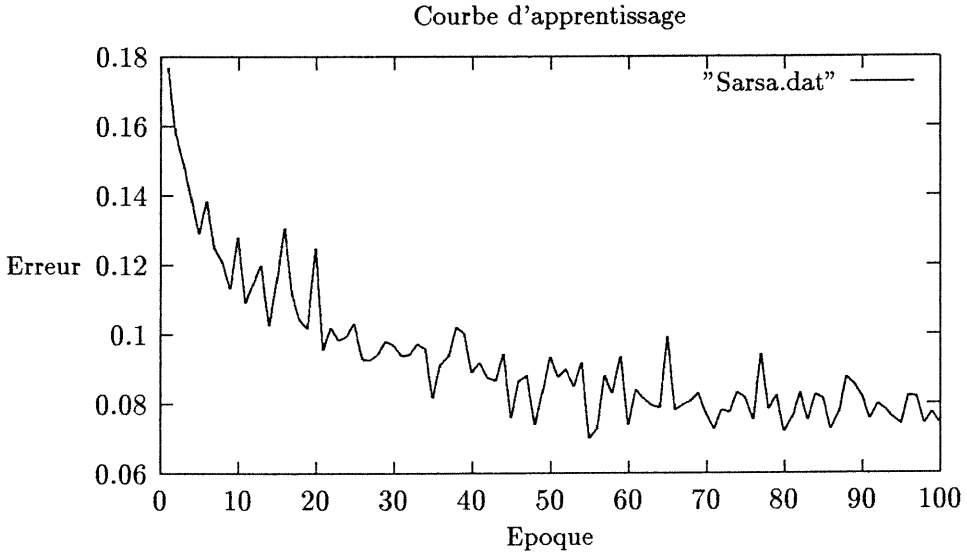


FIG. 7.7 – Erreur sur Q pour l'algorithme *Sarsa*

On remarque que l'erreur diminue surtout au cours des 25 premières époques. Elle continue de diminuer pendant les 25 époques suivantes. Par la suite, l'erreur commise atteint un plateau et sa valeur oscille autour de 0,08. L'erreur moyenne pour les 50 dernières est de 0,08081.

Cependant, nous remarquons rapidement que l'agent pour lequel l'erreur quadratique a la plus petite valeur numérique n'est pas nécessairement l'agent qui s'est le mieux comporté par rapport à l'objectif du jeu. Même

à l'inverse, l'agent ayant marqué le plus grand nombre de points au cours d'une partie est souvent l'agent dont l'erreur cumulative est la plus élevée. Ceci s'explique par le fait que le renforcement positif est obtenu uniquement quand un agent marque des points. Or, l'agent en possession du ballon obtient des points qu'une fois par seconde, une fréquence inférieure au rythme de ses décisions. Ce qui cause en bout de ligne des fluctuations dans les signaux de renforcement reçus. Les fonctions Q pour les actions choisies dans ces cas-là sont donc fondamentalement plus stochastiques et donc leurs approximations plus ardues.

Pour avoir une meilleure idée des performances que l'on peut espérer d'un agent suivant cet algorithme, nous l'opposons à trois agents utilisant l'heuristique et observons le pourcentage des points que l'agent adaptatif obtient. Des quatre agents entraînés, nous déterminons lequel est le plus performant en regardant le nombre de points que chacun a amassé au cours de la phase d'apprentissage. Nous faisons jouer à l'agent ainsi choisi et aux trois agents utilisant l'heuristique 100 parties au cours desquelles l'agent adaptatif ne fait aucun apprentissage. Voici les résultats ainsi obtenus :

Algorithme	Pourcentage des points accumulés	Écart-type
Sarsa(λ)	10,3885	1,7486

TAB. 7.1 – Performances Sarsa

L'écart-type est suffisamment petit pour que l'on puisse prendre en considération le résultat obtenu. Ce résultat est un peu décevant. Idéalement, l'agent adaptatif aurait dû obtenir au moins 25% des points. Cette valeur nous indiquerait que la stratégie suivie par l'agent adaptatif est au moins aussi efficace que l'heuristique, ce qui ne semble pas être le cas. Jusqu'ici, l'heuristique semble l'algorithme offrant les meilleures performances.

L'écart-type mesuré ici représente l'écart-type observé dans la moyenne des points obtenus par l'agent dont on mesure les performances en le faisant jouer 100 parties sans qu'il n'y ait d'apprentissage. La section 7.8.3 présente une mesure de l'écart-type obtenu quand la même expérience d'apprentissage est reprise plusieurs fois.

7.7 *Q-learning*

Nous présentons ici l'algorithme *Q-learning*, aussi appelé Q(λ) . Cet algorithme est un des algorithmes d'apprentissage par renforcement les plus répandus. Nous présentons ici que la version utilisant les traces remplaçantes. Les traces accumulantes peuvent y être intégrées de façon similaire à celle

vue pour l'algorithme Sarsa.

Initialiser $\vec{\theta}$ arbitrairement et $\vec{e} = \vec{0}$
Répéter (pour chaque épisode)
  Soit s, l'état initial de l'épisode
  Répéter (pour chaque pas de l'épisode)
    Selon une probabilité 1 - ε :
      a $\leftarrow argmax_a Q_t(a, s)$
      $\vec{e} \leftarrow \gamma\lambda\vec{e}$
    Sinon :
      a $\leftarrow$ action au hasard $\in A$(s)
      $\vec{e} \leftarrow \vec{0}$
    $\forall i \in F_a$
      e(i) $\leftarrow$ 1
    Prendre l'action a, observer la récompense r et l'état suivant s'
    $\delta \leftarrow r - Q_t(a, s)$
    Soit a' $\leftarrow argmax_a Q_{t+1}(a, s')$
    $\delta \leftarrow \delta + \gamma Q_{t+1}(a', s')$
    Calcul de $\Delta\vec{\theta}$. Pour chaque élément θ_i
      Calculer $\frac{\partial\delta}{\partial\theta_i}$
    $\theta_i \leftarrow \theta_i + \alpha\Delta\theta_i e_i$
  Jusqu'à ce que s' soit terminal

FIG. 7.8 – Algorithme *Q-Learning*

où encore une fois $\vec{\theta}$ est l'ensemble des paramètres à être appris, $\vec{e}$ est l'ensemble des traces d'eligibilité de ces paramètres, γ est un facteur d'atténuation temporel, λ est le pas d'apprentissage, δ la mesure de l'erreur et F_a est l'ensemble des traces d'éligibilité relatives à l'action a.

Encore une fois, le calcul des gradients s'effectue par rétro-propagation de façon à déterminer pour chaque unité son apport sur ce gradient.

7.7.1 Performances de l'algorithme *Q-Learning*

Nous procédons pour cet algorithme de la même façon que nous l'avons fait pour *Sarsa*. Nous entraînons quatre agents régis par l'algorithme *Q-Learning* en les opposant entre eux. Nous mesurons encore une fois la moyenne de l'erreur quadratique des quatre agents pour chaque époque.

La courbe d'apprentissage est très similaire à celle obtenue pour l'algorithme *Sarsa*. Encore une fois, l'erreur diminue graduellement au cours des 50 premières époques. L'erreur oscille également autour de 0,08 pour la deuxième moitié de la phase d'apprentissage. La valeur de l'erreur moyenne

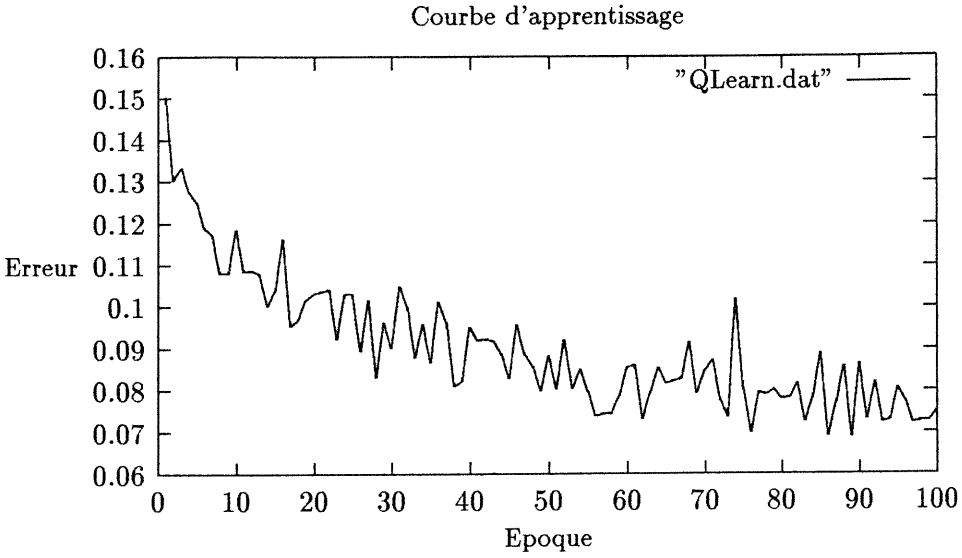


FIG. 7.9 – Erreur sur Q pour l'algorithme *Q-Learning*

des quatre agents pour les 50 dernières époques est de 0,07970, ce qui est inférieur à la mesure équivalente pour l'algorithme Sarsa, mais pas de façon significative.

Pour avoir une idée plus juste des performances de l'algorithme, nous reprenons l'évaluation par opposition à des copies de l'heuristique. Cette seconde évaluation donne également des résultats similaires à ceux obtenus pour l'algorithme Sarsa.

Algorithme	Pourcentage des points accumulés	Écart-type
Q-Learning	11,8791	1,7064

TAB. 7.2 – Performances Q-Learning

Bien que l'agent utilisant *Q-Learning* obtienne des performances supérieures à celles obtenues par *Sarsa*, l'intervalle de confiance observé ne nous permet de tirer des conclusions de cette différence. Et de plus, nous sommes encore loin du 25% recherché.

Nous cherchons donc à savoir si ce n'est qu'une simple question de capacité des réseaux. On se souvient que les courbes d'apprentissage des parties échantillonnées soulèvent des questions concernant les capacités des réseaux. Nous créons donc des agents utilisant l'algorithme *Q-Learning* dont la capacité des réseaux a été doublée. Pour s'assurer que cette augmentation de

capacité puisse avoir une influence, la période d'entraînement est également doublée. Nous avons ainsi obtenu les résultats suivant :

Algorithme	Pourcentage des points accumulés	Écart-type
Q(λ) amélioré	12,1022	2,1247

TAB. 7.3 – Performances Q-Learning amélioré

Les performances obtenues par cet agent semble un peu supérieures à celles obtenues par l'agent dont les réseaux ont la moitié de la capacité. Mais encore une fois, la valeur de l'écart-type est plus grande que la différence observée. On peut donc difficilement conclure que l'augmentation de la capacité a une influence directe sur les performances de l'agent.

7.8 Descente de gradient sur la politique

Sutton *et al.* ont présenté au cours de l'année dernière une nouvelle famille d'algorithmes [93] qui apprennent l'ensemble de la politique et non simplement les fonctions d'évaluation des options de façon individuelle ou encore la fonction d'évaluation de l'état et les probabilités de transitions. L'apprentissage se fait d'une part sur les fonctions d'évaluation des options, mais aussi sur la probabilité avec laquelle une option devrait être choisie. Cette probabilité dépend non seulement de l'évaluation qui en est faite, mais aussi des évaluations des autres options.

Cet algorithme s'inspire d'une part de *Q-learning*, mais encore d'avantage de la variation proposée par Baird [9]. Plutôt que d'apprendre les fonctions Q de chaque option, l'algorithme présenté par Baird apprend ce qu'il appelle la fonction d'avantage, c'est-à-dire l'avantage que peut potentiellement apporter le choix d'une option par rapport à la moyenne des options. Dans le cas de l'algorithme de Sutton *et al.*, trois fonctions fortement interdépendantes doivent être apprises.

Tout d'abord, le but premier de cet algorithme est d'apprendre, pour un état donné, les probabilités avec lesquelles chaque option devrait être choisie. On calcule ces probabilités par la fonction P suivante :

$$P(a,s) = \exp(G(a,s)) / \sum_b \exp(G(b,s)) \tag{7.5}$$

où G est la fonction qui doit être apprise. G peut être simplement

une combinaison linéaire de paramètres appris ($\vec{\theta}$) et d'un vecteur de caractéristiques ($\vec{F}(a,s)$) propre à chaque couple état-action :

$$G(a,s) = \sum_i \theta_i F_i(a,s) \tag{7.6}$$

Cependant, on obtient un apprentissage nettement supérieur à l'aide d'un réseau de neurones que par une simple combinaison linéaire de paramètres libres. Ce réseau reçoit en entrée ce même vecteur ($\vec{F}(a,s)$). Comme vecteur d'entrée, nous utilisons simplement la concaténation des différents vecteurs d'entrées développés pour évaluer les options individuellement. Pour une action donnée, on affecte à la section du vecteur correspondant à cette option les valeurs des variables d'état qui avaient été retenues comme pertinentes et 0 aux autres valeurs. Si un tel vecteur est utilisé, c'est que le vecteur $\vec{F}(a,s)$ doit avoir la même longueur pour toutes les valeurs possibles de a.

L'apprentissage de la fonction G nécessite la fonction Q afin d'estimer l'erreur commise par le réseau :

$$\Delta\theta = Q(a_t, s_t) * \frac{\partial P(a_t, s_t)}{\partial \theta} \tag{7.7}$$

Ici, la fonction Q n'a pas tout à fait la même signification que celle qu'on lui prête habituellement, i.e. celle de l'algorithme de *Q-Learning*. La fonction Q représente l'avantage du choix d'une action dans une situation, comme dans la fonction A d'avantage telle que décrite par Baird [9], $A(a,s)^\pi = Q(a,s)^\pi - V(s)^\pi$. Encore une fois, cette fonction Q pourrait se calculer par une combinaison linéaire de la façon suivante :

$$Q(a,s) = \sum_i W_i(f_i(a,s) - \sum_b P(b,s)F_i(b,s)) \tag{7.8}$$

Par contre, il est préférable de tirer profit de la non-linéarité que permet l'utilisation de réseaux de neurones. Un réseau sera donc employé pour apprendre la fonction Q. Ce réseau se voit fournir en entrée le vecteur $\vec{F}(a,s) - \sum_b P(b,s)\vec{F}(b,s)$. Ici, il apparaît clairement que le vecteur $\vec{F}(a,s)$ doit avoir la même longueur pour toutes les valeurs de a.

Cependant, l'apprentissage de la fonction nécessite la fonction V, puisque la fonction Q se veut l'avantage d'une option par rapport à cette fonction V.

$$\Delta W = (Q(a_t, s_t) - V(s_t)) * \frac{\partial Q(a_t, s_t)}{\partial W} \tag{7.9}$$

Il faut donc un troisième réseau pour apprendre la fonction V. Soit Z, l'ensemble des paramètres libres de ce réseau. L'équation linéaire correspondante est :

$$V(s) = \sum_i Z_i s_i \tag{7.10}$$

Cependant, une fois de plus un réseau de neurone semble mieux adapté qu'un ensemble de paramètres libres combinés linéairement. La règle d'apprentissage pour ce réseau est alors :

$$\Delta Z = (V(s_t) - \gamma V(s_{t+1}) - r_{t+1}) \frac{\partial V(s_t)}{\partial Z} \tag{7.11}$$

7.8.1 Entraînement des réseaux

Des tests préliminaires nous ont permis de remarquer qu'il est très avantageux de ne pas utiliser de couches cachées dans les différents réseaux de neurones. En effet, d'une part l'entraînement se fait beaucoup plus rapidement et d'autre part les résultats sont équivalents sinon supérieurs.

Le premier essai consiste à entraîner les trois réseaux en parallèle. Cependant, nous observons d'étranges résultats en procédant de la sorte. Les réseaux s'emballent de sorte que l'évaluation du réseau G sur les probabilités que devraient avoir certaines options prennent des valeurs démesurées ; par conséquent ces options sont toujours choisies, bien que les résultats découlant de ces choix soient décevants. On remarque qu'un seul agent obtient pratiquement la totalité des points disponibles pour chaque partie.

Nous avons choisi d'entraîner en alternance les réseaux dédiés à l'apprentissage des fonctions V et Q individuellement, en commençant par la fonction V. Comme l'indique l'équation 7.9, l'apprentissage de la fonction Q dépend directement de la fonction V. Si la fonction V évolue sans cesse, l'approximation de la fonction Q fait alors face au problème de la cible mouvante. C'est pourquoi nous avons choisi d'entraîner ces deux fonctions individuellement.

Dans un premier temps, chaque réseau sera entraîné par périodes de 5 époques où chacune des époques d'entraînement correspond à une partie de cinq minutes. Pour sa part, le réseau dédié à l'apprentissage de la fonction G ne semblait pas être affecté de façon indésirable par l'évolution des deux autres réseaux. Son apprentissage se poursuivra donc tout au long de la procédure, aussi bien pendant l'apprentissage de la fonction V que celui de la fonction Q.

7.8.2 Performances de l'algorithme de Sutton

Nous avons comparé les deux programmes d'entraînement. Dans la première les fonctions V et Q sont apprises en alternance, avec des périodes de 5 parties de cinq minutes. La seconde consiste à apprendre toutes les fonctions en parallèle.

Pour chacune de ces deux méthodes, nous procédons comme nous l'avons fait pour les deux algorithmes précédents. Nous entraînons quatre agents en les faisant jouer entre eux. De ces quatre agents, le plus performant est déterminé en observant les résultats obtenus lors de la phase d'apprentissage.

Afin de pouvoir comparer les résultats obtenus par les agents utilisant cet algorithme avec ceux obtenus par les agents utilisant les autres algorithmes, nous opposons l'agent choisi comme étant le plus performant à trois agents régis par l'heuristique pour 100 parties au cours desquelles aucun apprentissage n'est effectué. Puisque les fonctions apprises par cet algorithme n'ont pas tout à fait la même signification que celles apprises dans les deux autres algorithmes, aucune comparaison ne peut être faite par rapport à l'erreur commise.

Nous présentons ici la courbe de l'évolution de l'agent ayant appris les fonctions V et Q en alternance.

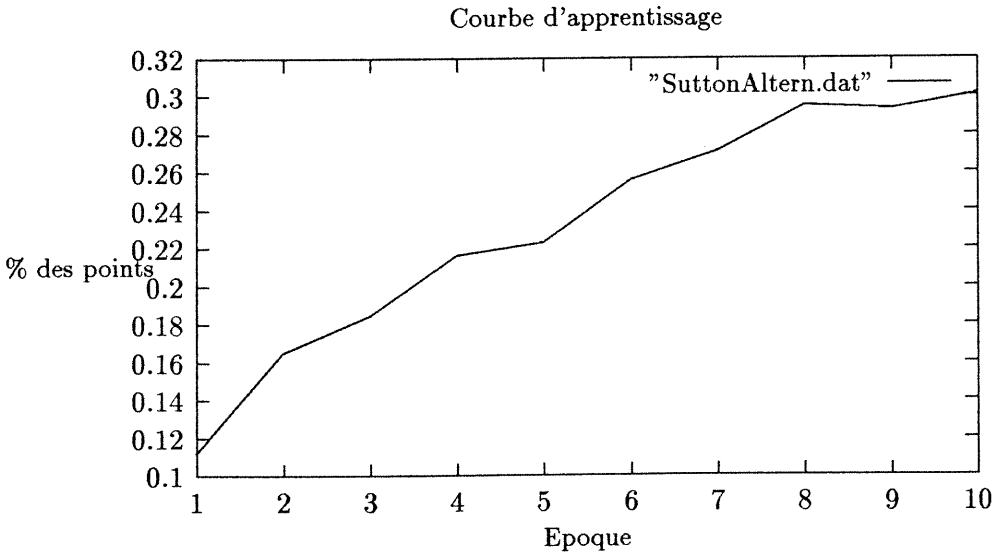


FIG. 7.10 – Apprentissage Sutton, V et Q en alternance

Comme il a déjà été mentionné, l'apprentissage se fait beaucoup plus ra-

pidement que pour les deux autres algorithmes. La partie la plus importante de l'apprentissage se fait au cours des dix premières parties. L'alternance dans la fonction apprise se fait à l'épisode cinq. Cependant, on voit très peu de différence dans la courbe d'apprentissage à ce point. On peut en conclure que l'apprentissage de chaque fonction a le même impact sur les performances de l'agent. L'absence de couche cachée dans les réseaux de neurones explique cette différence dans la période d'entraînement nécessaire. Comme nous le verrons plus loin, ce même algorithme requiert des périodes d'entraînement comparables à celle des deux autres algorithmes si les réseaux comportent des couches cachées.

Nous présentons à la figure 7.11 la courbe d'apprentissage de l'agent pour lequel l'apprentissage des fonctions V et Q se déroule en parallèle à titre indicatif. On remarque évidemment qu'il n'y a pas de coupure à l'épisode cinq puisqu'il n'y a pas de changement dans la nature des fonctions apprises.

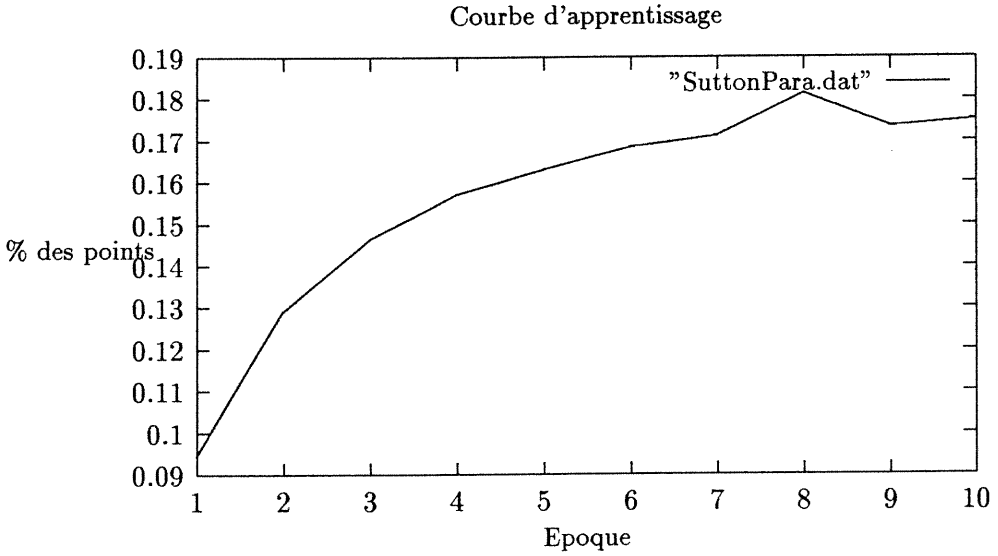


FIG. 7.11 – Apprentissage Sutton, V et Q en parallèle

Apprentissage	Pourcentage des points accumulés	Écart-type
V et Q en parallèle	17,4944	3,0344
V et Q en alternance	30,1226	3,6550

TAB. 7.4 – Performances Sutton

Il apparaît clairement qu'il est préférable d'entraîner les réseaux apprenant les fonctions V et Q en alternance plutôt qu'en parallèle. L'agent ayant

appris ces deux fonctions une à la suite de l'autre s'est d'ailleurs hissé au-dessus de la barre des 25%.

7.8.3 Intervalle de confiance

Comme nous l'avons déjà mentionné, l'évaluation de chaque essai se fait en mesurant les performances de l'agent qui semble le plus prometteur obtenu à la fin de la phase d'entraînement. Il est légitime de se demander jusqu'à quel point on peut avoir confiance dans cette méthode d'évaluation.

Afin de mesurer cet intervalle de confiance, nous nous attardons à une des expériences qui ont mené aux résultats présentés à la section 7.8.2, soit l'entraînement d'agent par l'algorithme de Sutton où l'apprentissage des fonctions V et Q se fait en alternance.

Nous reprenons donc précisément cette même expérience à dix reprises. Nous calculons pour ces dix essais la moyenne des résultats obtenus et surtout l'écart-type des résultats par rapport à cette moyenne.

Moyenne des résultats pour dix essais	Écart-type
31,0881	1,5143

TAB. 7.5 – Intervalle de confiance

Pour ces dix essais, l'écart-type observé est de 1,5143. Les écarts-type qui accompagnent les autres résultats présentés ne tiennent pas compte de cette variation que l'on observe sur les résultats de deux expériences identiques. On doit donc additionner le carré de 1,5143 au carré des valeurs observées, puis extraire la racine carrée du résultat pour obtenir la réelle valeur de l'écart-type, selon l'équation $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$.

7.9 Comparaison des trois algorithmes

Nous voulons comparer entre eux les trois algorithmes à l'essai et déterminer hors de tout doute lequel est le mieux adapté à la tâche qui nous intéresse. Les mesures de performances obtenues en opposant un agent à trois agents utilisant l'heuristique nous donne déjà une première idée de la pertinence de chaque algorithme. Cependant, rien ne nous assure qu'une stratégie efficace contre un agent utilisant l'heuristique comme politique est efficace contre d'autres politiques.

Nous voulons donc opposer directement entre eux les agents utilisant les différents algorithmes mis à l'essai jusqu'ici. Nous jetons dans l'arène un

représentant de chacun des trois algorithmes, soit l'agent le plus performant obtenu pour chacun des trois algorithmes. Nous ajoutons un agent utilisant l'heuristique pour, d'une part, compléter le quatuor, et d'autre part, faire office de point de référence. Pour justifier l'utilisation d'un algorithme d'apprentissage, l'agent l'utilisant doit être en mesure de battre un agent utilisant l'heuristique.

Algorithmes	Pourcentage des points accumulés	Écart-type
Heuristique	34,747	2,209
Sarsa(λ)	8,915	1,043
Sutton	46,562	2,627
Q(λ)	9,776	0,998

TAB. 7.6 – Comparaison des trois algorithmes

Nous avons calculé les écarts-type des résultats obtenus. Ces écarts sont largement inférieurs aux valeurs des résultats. Il est donc permis de penser que ces résultats sont significatifs.

L'agent utilisant l'algorithme de descente de gradient sur la politique sort grand gagnant de cette compétition. Il a accumulé un peu moins de la moitié du total des points disponibles. C'est d'ailleurs le seul agent adaptatif qui a fait mieux que celui utilisant l'heuristique. On remarque que les résultats obtenus par les algorithmes Sarsa(λ) et Q(λ) sont très similaires. Ceci s'explique très bien par le fait que ces deux algorithmes comportent beaucoup de similarités. Cependant, l'agent utilisant Q(λ) a légèrement mieux accompli la tâche. L'algorithme de descente de gradient sur la politique proposé par Sutton *et al.* est donc retenu comme le mieux adapté pour le problème qui nous intéresse.

Le succès de cet algorithme face aux deux autres peut s'expliquer de deux façons. Tout d'abord, un seul réseau apprend la fonction Q. Ce réseau se voit donc fournir à l'entrée l'ensemble des variables d'état. Cette information est présentée de façon à suggérer quelles sont les informations pertinentes pour chacune des décisions. Mais il peut quand même se produire un certain partage des poids entre les options. Puisqu'un seul réseau apprend la fonction Q pour chacune des options, il peut y avoir des généralisations dans les patrons de connexions entre les unitées qui s'appliquent à l'évaluation de plusieurs options. Un sous-ensemble du réseau peut effectuer un calcul qui est utile dans l'évaluation de plus d'une option.

Ensuite, et c'est d'ailleurs là la principale différence entre cet algorithme et les deux autres, un réseau est entièrement dédié à l'apprentissage de la fonction V. Cette fonction n'étant qu'une évaluation de l'état actuel sans aucune considération pour les différentes options, peut-être est-elle plus simple

à apprendre. D'ailleurs, l'erreur commise sur les approximations de cette fonction sont presque d'un ordre de grandeur inférieur à l'erreur commise sur l'approximation de la fonction Q. Puisque l'ensemble du processus de prise de décisions repose sur cette fonction, une bonne approximation de celle-ci constitue une excellente base pour le reste.

7.9.1 Partage des poids

Afin de vérifier ces hypothèses, nous avons bâti un agent entraîné avec *Q-Learning*, mais qui utilise un seul réseau à plusieurs sorties pour les fonctions Q à apprendre. On a pu ainsi vérifier s'il y a gain à obtenir dans le partage des poids dans la couche d'entrée. Encore une fois, nous avons entraîné quatre agents reposant sur cette architecture et nous avons retenu le meilleur des quatre. Nous l'avons ensuite opposé à un agent utilisant l'algorithme de Sutton, un agent utilisant l'heuristique et finalement un agent utilisant l'algorithme *Q-Learning*, mais qui emploie plusieurs réseaux pour apprendre les fonctions Q. Ce dernier agent permettra d'observer plus précisément le gain obtenu par le partage des poids, si un tel gain existe.

Algorithme	Pourcentage des points accumulés	Écart-type
Q(λ), un seul réseau	11,0881	1,8012

TAB. 7.7 – Performances Q-Learning, un seul réseau

Nous refaisons une série de parties afin de comparer les algorithmes en les opposant directement les uns aux autres. Nous insérons dans la partie deux agents utilisant l'algorithme *Q-Learning*. Un de ces agents comporte un seul réseau de neurones alors que l'autre en utilise un différent pour chaque action.

Algorithmes	Pourcentage des points accumulés	Écart-type
Q(λ), un seul réseau	12,29	1,4739
Sutton	41,649	2,2079
Heuristique	31,712	1,7679
Q(λ), plusieurs réseaux	14,349	1,1242

L'agent *Q-Learning* utilisant un seul réseau a accumulé moins de points que l'agent qui en utilise plusieurs. Il apparaît clairement que l'utilisation d'un seul réseau n'offre pas de gain de performance, ou du moins que plusieurs réseaux spécialisés avec chacun un nombre d'entrées restreints procurent de meilleurs résultats. Mais l'agent utilisant l'algorithme de Sutton demeure le grand gagnant. Nous pouvons donc penser que la supériorité de cet algorithme dans l'apprentissage de la tâche qui nous intéresse provient de l'utilisation d'un réseau dédié à l'apprentissage de la fonction V.

7.10 Architectures des réseaux

Il faut maintenant déterminer quelle est l'architecture la mieux adaptée pour les réseaux de neurones employés.

L'algorithme retenu n'est pas très souple pour ce qui est du nombre de réseaux utilisés. Trois fonctions doivent être apprises et le nombre de paramètres fournis à l'entrée doit être le même pour deux de ces trois fonctions. Par contre, il n'en demeure pas moins qu'il reste certains aspects qui doivent être fixés.

La question principale à laquelle une réponse doit être trouvée concerne la pertinence de l'utilisation de couches cachées dans les différents réseaux de neurones. Ensuite, le nombre optimal d'unités dans ces couches cachées aurait du faire l'objet d'une étude, cependant, comme nous le verrons, l'utilisation d'une couche cachée n'apporte aucun gain.

7.10.1 Couches cachées

Jusqu'ici, tous les résultats concernant l'algorithme sélectionné ont été obtenus par des agents ne comportant pas de couches cachées dans les différents réseaux de neurones utilisés dans l'algorithme. Nous présentons donc ici les performances obtenues par des agents qui utilisent l'algorithme de Sutton et dont les réseaux de neurones comprennent des couches cachées.

La première observation est que le nombre d'époques nécessaire à l'entraînement est beaucoup plus proche de ceux observé pour les autres algorithmes. Le temps d'apprentissage est donc beaucoup plus influencé par la présence d'une couche cachée que par l'algorithme employé. Ensuite, nous reprenons l'idée de l'apprentissage des fonctions V et Q en alternance. Cependant, nous l'adaptons à ce temps d'apprentissage qui est maintenant plus long en alternant la fonction apprise à chaques 50 époques.

Suite à la phase d'apprentissage, nous sommes maintenant à la phase d'évaluation. Une fois de plus, nous opposons l'agent le plus performant obtenu lors de la phase d'apprentissage à trois agents utilisant l'heuristique comme politique. Nous les faisons jouer 100 parties et observons le pourcentage des points que l'agent adaptatif a récolté.

Durée des phases d'entraînement	Pourcentage des points accumulés	Écart-type
50 époques	26,474	2,7443

TAB. 7.8 – Performances Sutton, avec couche cachée, phases de 50 époques

Nous remarquons que cet agent obtient des résultats inférieurs à ceux des agents ne comportant pas de couches cachées. Il se hisse tout juste au-dessus de la barre des 25 %. Cependant, l'écart-type est plus grand que la différence entre le résultat obtenu et cet objectif. De plus, le temps d'apprentissage nécessaire pour obtenir ces performances est plus élevé d'un ordre de grandeur.

Pour s'assurer de validité des résultats obtenus, nous reprenons la même expérience, mais en alternant la fonction apprise à toutes les 25 époques plutôt que 50.

Durée des phases d'entraînement	Pourcentage des points accumulés	Écart-type
25 époques	22,8487	2,2180

TAB. 7.9 – Performances Sutton, avec couche cachée, phases de 25 époques

L'agent obtient moins de points que celui dont l'alternance de la fonction apprise se produit aux 50 époques. Nous en venons donc à la conclusion qu'il est préférable de ne pas utiliser de couche cachée dans les réseaux de neurones.

Aucune expérience n'a été réalisée avec les algorithmes Q(λ) et Sarsa(λ) en utilisant des réseaux sans couche cachée. Le gain de performance observé pour l'algorithme de Sutton nous porte à croire que le gain que l'on pourrait ainsi obtenir pour ces algorithmes ne serait pas suffisant pour que l'un ou l'autre détrône l'algorithme de Sutton. On pourrait cependant s'en assurer de façon expérimentale.

7.10.2 Capacité des réseaux

Les résultats précédants indiquent que l'utilisation de couches cachées dans les réseaux de neurones n'apportent pas de gain de performance et même qu'au contraire, la présence d'une couche abaisse la qualité des performances des agents. Cependant, les expériences qui ont mené à cette conclusion ont été réalisées avec une capacité précise pour les réseaux avec couches cachées. Le nombre d'unités sur la couche cachée caractérise la capacité du réseau.

Nous voulons savoir si en réduisant la capacité des réseaux, donc en réduisant la quantité d'unités cachées, les performances des agents s'en trouvent améliorées, voir supérieures à celles des agents ne comprenant pas de couches cachées. Jusqu'ici, les couches cachées disposaient d'un nombre d'unités correspondant au double du nombre d'unités sur la couche d'entrée. Nous réduisons donc ce nombre de moitié et reprenons l'expérience dans les

même conditions.

Capacité	Pourcentage des points accumulés	Écart-type
Capacité réduite	23,4551	2,2546

TAB. 7.10 – Performances Sutton, couche cachée avec capacité réduite

Les résultats obtenus sont très comparables à ceux obtenus auparavant par des agents disposant de couches cachées. En fait, les résultats sont légèrement inférieurs à ceux obtenus par l'agent de la section 7.10.1. Cependant, l'intervalle de confiance ne permet de tirer de conclusion précise autre que le fait que l'utilisation de couche cachée n'apporte aucun gain.

7.11 Programme d'entraînement

Maintenant que le choix de l'algorithme s'est arrêté à l'algorithme de descente de gradient sur la politique et que nous avons déterminé l'architecture de réseaux de neurones offrant les meilleurs résultats, il faut à présent déterminer la façon la plus efficace de les entraîner.

Nous savons déjà qu'il est préférable d'entraîner individuellement les réseaux destinés à l'apprentissage des fonctions V et Q. Dans un premier temps, nous voulons à présent déterminer si l'on peut pousser plus loin l'apprentissage de l'agent si le pas d'apprentissage est diminué graduellement. Ensuite, deux hypothèses de techniques agissant comme catalyseur de l'apprentissage doivent être mises à mettre à l'essai. Nous voulons savoir s'il existe une façon d'utiliser l'heuristique comme entraîneur. Nous voulons également explorer la possibilité de partager l'apprentissage fait par plusieurs agents simultanément. Pour vérifier ces hypothèses, nous observons le comportement d'agents entraînés en utilisant ces techniques.

7.11.1 Pas d'apprentissage

Toutes les modifications qui sont apportées au poids des connexions entre les unités des réseaux de neurones dans la direction du gradient sont ajustées par un facteur multiplicatif. On appelle ce facteur le pas d'apprentissage. Généralement, la valeur du pas d'apprentissage est réduite graduellement au cours de l'apprentissage. Plus l'entraînement avance, plus les modifications à apporter aux poids sont en fait de fins ajustements. Les premières modifications apportées aux valeurs des poids cherchent à les amener dans des plages de valeurs voisines des valeurs théoriquement optimales. Par la

suite, les ajustements fins servent à atteindre par raffinements successifs ces mêmes valeurs théoriquement optimales.

Nous voulons donc savoir si ce principe général s'applique à notre cas. D'une part, nous voulons savoir si les performances des agents continuent de s'améliorer si nous leur faisons subir des phases apprentissage supplémentaires. Nous voulons également découvrir à quel point on peut espérer pousser l'apprentissage.

Nous reprenons donc les agents décris à la section 7.8.1 pour lesquels l'apprentissage des fonctions V et Q s'est déroulé en alternance. Ces agents sont les plus performants obtenus jusqu'ici. Ces agents ont été entraînés 10 époques avec un pas d'apprentissage d'une valeur de 0,001. Les résultats présentés dans le premier tableau sont repris de la section 7.8.1.

Pas d'apprentissage	Pourcentage des points accumulés	Écart-type
0,001	30,1226	3,6550

TAB. 7.11 – Performances Sutton, Lambda=0,001

Nous voulons pousser plus loin l'apprentissage pour voir si les performances continuent de s'améliorer. Nous divisons donc la valeur du pas d'apprentissage par trois. Puisque ce pas est plus petit, un plus grand nombre d'époques d'entraînement est nécessaire pour qu'une différence soit observable. Nous entraînons donc les agents pendant 20 époques plutôt que 10 et en alternant la fonction apprise au bout de 10 époques plutôt que cinq. Suite à cette phase d'apprentissage, nous procédons à une phase d'évaluation afin de constater les résultats.

Pas d'apprentissage	Pourcentage des points accumulés	Écart-type
0,000333	33,3409	4,0789

TAB. 7.12 – Performances Sutton, Lambda=0,000333

Les performances de l'agent semble s'être légèrement améliorées. Cependant, la valeur de l'écart-type est supérieure à cette amélioration. Nous poussons l'idée plus loin en utilisant un pas d'apprentissage 10 fois plus petit que la valeur originale. Nous gardons la même durée pour la période d'apprentissage.

Pas d'apprentissage	Pourcentage des points accumulés	Écart-type
0,0001	34,7868	4,1640

TAB. 7.13 – Performances Sutton, Lambda=0,0001

Encore une fois, l'agent semble un peu plus performant. Cependant, l'écart-type est beaucoup plus important que le gain obtenu. Cependant, si on compare ce dernier résultat au premier, il semble réellement se dégager une tendance qui permet de croire que l'apprentissage se poursuit réelement.

7.11.2 L'heuristique comme entraîneur

Au tout début de leur apprentissage, les agents ne sont pas très performants. Il faut quelques époques d'entraînements avant d'observer des comportements qui puissent sembler un tant soit peu intelligents. Jusqu'ici, toutes les phases d'entraînement mettaient en cause des agents qui commençaient tous de ce point de départ. Ce qui veut dire qu'au début de son apprentissage, un agent est opposé à d'autres agents dont la qualité du jeu n'est pas très relevée. Les stratégies qu'un agent doit développer pour battre ces agents peu menaçants sont donc simples. Puisque ses adversaires évoluent, l'agent devra par la suite raffiner ses stratégies. Mais au départ, des stratégies très simples peuvent suffire. Peut-être qu'en opposant un agent en situation d'apprentissage dès le début de son entraînement à des agents utilisant certaines formes de stratégies plus évoluées pourra stimuler son apprentissage. Nous opposerons un agent à des agents utilisant l'heuristique pour découvrir si cet agent sera par la suite plus performant que des agents soumis à des expériences d'apprentissage différentes.

7.11.3 Redistribution des poids

Dans toutes les expériences que nous avons réalisées, chaque agent apprenait individuellement et les performances de chacun étaient mesurées une fois l'apprentissage complété. Un agent dont l'apprentissage était plus lent au début demeurait inférieur aux autres agents tout au long de l'entraînement. Une grande partie de l'apprentissage utile s'effectue lorsqu'un agent prend possession du ballon et le conserve. C'est seulement à ce moment que l'agent obtient de l'information sur les situations qui peuvent mener à l'obtention du ballon et sur les décisions à prendre dans ces situations. Un agent moins performant éprouvera beaucoup de difficultés à obtenir possession du ballon, alors que les autres deviendront de plus en plus efficaces dans cette tâche. Les chances pour cet agent de s'améliorer s'en trouveront par le fait même réduites. L'hypothèse, c'est qu'il existe la possibilité d'un gain à obtenir en partageant entre les agents l'expérience acquise par chacun d'entre eux. Ainsi, un agent qui connaît un mauvais départ pourra néanmoins être utile pour l'ensemble de la phase d'apprentissage.

La façon dont est envisagée le partage de l'expérience est très simple.

Après chaque partie, l'agent le plus performant est sélectionné. Le critère de performance est une fois de plus le pourcentage des points obtenus pour la durée de la partie. L'ensemble des paramètres des réseaux utilisés par cet agent est redistribué aux autres agents. Ainsi, après chaque partie, seul l'agent le plus performant est retenu et quatre copies de celui-ci sont remises dans l'arène pour poursuivre l'entraînement.

7.11.4 Résultats

Nous voulons mettre sur pied des programmes d'entraînement permettant de vérifier conjointement ces deux hypothèses. La première option est de simplement entraîner un agent en le faisant jouer contre trois copies de l'heuristique. Une autre alternative consiste à faire jouer entre eux quatre agents en situation d'apprentissage et de redistribuer les poids des réseaux de l'agent le plus performant aux trois autres. Finalement, la troisième option est en quelque sorte un compromis entre les deux précédentes. Il s'agit d'entraîner deux agents en les faisant joueur contre deux autres agents utilisant l'heuristique pour diriger leurs actions. Encore une fois, après chaque époque, le plus performant des deux agents apprenant est sélectionné et ses poids sont copiés à l'autre agent adaptatif.

Pour comparer les résultats obtenus, nous évaluons individuellement les agents obtenus par chacune des trois procédures en les opposant une fois de plus à trois agents pour lesquels l'heuristique est la politique suivie.

Méthode	Pourcentage des points accumulés	Écart-type
Un seul agent	10,4044	2,6147
2 agents, poids redistribués	14,6294	2,3625
4 agents, poids redistribués	31,4926	4,1634

TAB. 7.14 – Performances des programme d'entraînement

Les résultats démontrent clairement que l'utilisation de l'heuristique comme entraîneur n'apporte aucun gain. Au contraire, l'agent dont l'entraînement se déroule contre trois agents utilisant l'heuristique se comporte moins bien que les agents obtenus précédemment. Cependant, il faut prendre en considération un autre facteur. L'agent qui a appris en jouant contre trois copies de l'heuristique est seul à apprendre et est automatiquement celui dont les performances sont évaluées. Dans les phases d'entraînement vues auparavant, l'agent ayant accumulé le plus de points était choisi parmi quatre agents. Il y avait donc déjà là une présélection. Ensuite, le niveau de jeu des agents utilisant l'heuristique est constant. Une fois qu'un agent adaptatif a appris à les battre, son apprentissage atteint un plafond. Un

agent apprenant en jouant contre d'autres agents adaptatifs fait toujours face à de nouveaux défis.

Le partage de l'expérience, du moins la façon dont nous l'appliquons, ne semble pas plus amener de gains. Les résultats de l'agent obtenu en redistribuant les poids aux quatre agents à la fin de chaque époque sont très similaires à ceux obtenus précédemment. En fait, l'écart-type est largement supérieur à la différence de sorte que nous ne pouvons pas conclure qu'on puisse ainsi obtenir un gain.

Cette dernière expérience clot la démarche suivie. Le chapitre suivant résume l'ensemble des résultats obtenus et présente des directions vers lesquelles pourraient aller des travaux futurs.

Chapitre 8

Conclusion

Pour terminer, nous présentons tout d'abord la solution finale retenue concernant le problème qui nous intéresse. Ensuite, nous voyons comment l'expérience d'apprentissage des agents peut être poussée et les améliorations qu'on pourrait apporter à ceux-ci. Enfin, nous discutons des généralisations possibles qui peuvent être tirées de l'étude que nous avons fait.

8.1 Solution retenue

Le problème qui est posé a été solutionné grâce aux techniques d'apprentissage par renforcement. Ces algorithmes présentent l'avantage d'apprendre une tâche avec comme simple aide extérieure un simple signal indiquant la souhaitabilitée de la derniére décision. De plus, il existe des mécanismes permettant une certaine souplesse quant à la temporalité des récompenses reçues. Cependant, pour que le système converge dans des délais raisonnables, une certaine connaissance du domaine de la tâche doit être incorporé à l'agent par le concepteur. La nature de ces connaissances et la façon de les intégrer peuvent avoir de très grandes répercution sur le temps d'apprentissage de l'agent.

L'architecture de base conçue répond parfaitement aux besoins de notre problème. Elle permet une certaine souplesse quant à l'algorithme de prise de décision utilisé. Au cours du projet, nous avons eu à bâtir des agents réalisant des tâches plus simples que la tâche décrite dans cet ouvrage. Ces agents n'effectuaient aucun apprentissage. La même architecture de base a pu être employée dans la conception de ces agents. Le module de planification de trajectoire, qui repose principalement sur l'algorithme A^*, est à la fois simple et efficace.

Parmi les différents algorithmes d'apprentissage que nous avons étudiés, l'algorithme de descente de gradient sur la politique proposé par Sutton *et al.* offre les meilleures performances dans le contexte de notre problème. Il a été déterminé qu'il est préférable d'entraîner en alternance deux des approximateurs de fonctions que comporte cet algorithme, soit les approximateurs des fonctions V et Q. Les agents n'utilisant aucune couche cachée dans leurs réseaux de neurones se sont avérés aussi performants, sinon plus, que les agents munis de couches cachées, peu importe le nombre d'unités présentes sur les couches cachées. L'apprentissage peut être poussé si le pas d'apprentissage est diminué graduellement.

Deux hypothèses avaient été émises concernant des techniques pour accélérer l'apprentissage des agents. D'une part l'heuristique pouvait être utilisée en guise d'entraîneur pour les agents adaptatifs. D'autre part l'expérience acquise par chaque agent pouvait être partagée entre ceux-ci. Aucune de ces deux techniques, ou du moins l'utilisation que nous en avons faite, n'a donné de meilleurs résultats que l'entraînement parallèle d'agents indépendants.

8.2 Apprentissage supplémentaire

Il serait possible de poursuivre plus loin l'exploration des possibilités concernant les sous-ensembles des variables d'état jugés pertinents à l'apprentissage d'une fonction. Nous avons fait des choix à ce sujet au meilleur de notre connaissance, en nous basant entre autres sur l'heuristique développée et des observations empiriques. Cependant, une étude comparative entre différentes possibilités aurait pu donner lieu à une solution finale encore plus performante.

À l'intérieur de notre système, certains choix d'importance secondaire se font au moyen de fonctions déterministes ou encore par des fonctions ayant des composantes stochastiques, mais pour lesquelles aucun apprentissage n'est fait. C'est notamment le cas du module de planification de trajectoire. Les terrains sur lesquels se déroule le jeu, bien qu'ils soient nombreux, sont en quantité limitée. Les agents pourraient apprendre des trucs spécifiques à chacun : les endroits qui sont généralement dangereux, les endroits où les autres joueurs tentent de se cacher , etc.

C'est également le cas de la fonction qui détermine la quantité de *Chi* qui devrait être affectée à l'utilisation d'une habileté spéciale. La fonction apprise pourrait avoir en entrée, en plus des paramètres ayant mené à ce choix, l'évaluation de la fonction Q de l'action correspondante. On peut penser que plus la valeur de la fonction Q est grande, plus on a avantage à affecter une grande quantité de *Chi* à l'utilisation de l'habileté spéciale. Ceci

est particulièrement vrai dans le cas des habiletés offensives.

Deux architectures de réseaux de neurones ont été mises à l'essai au cours du projet : des réseaux avec couche cachée et des réseaux sans couche cachée. L'utilisation de couche cachée n'a pas semblée avoir d'utilitée dans le cas présent. Cependant, il est concevable que cette couche cachée aurait pus avoir un impact bénéfique sur les performances si des connexions directes entre les unitées d'entrée et les unitées de sortie avaient été ajoutées. On aurrait ainsi potentiellement eu la rapidité de convergeance des réseaux sans couche caché et la subtilité de réponses des réseaux qui en possédent une.

Des expériences concernant les algorithmes sarsa(λ) et Q(λ) sans couche cachée ou encore avec une architecture hybride impliquant une couche cachée et des connexions directes entrée-sortie aurait pu renforcir la démonstration que l'algorithme de Sutton est le mieux adpaté pour notre cas.

La façon dont avons tenté de partager l'expérience entre les agents n'a provoqué aucun gain dans leurs performances. Mais peut-être qu'un gain est possible en partageant l'expérience. Plutôt que de redistribuer les poids après chaque époque d'entraînement, il serait peut-être plus profitable de partager l'expérience à chaque cinq, dix ou 25 époques. Ainsi l'agent dont les poids sont distribués aura eu la chance d'expérimenter toutes les facettes du jeu.

8.3 Généralisation

Une façon de valider les résultats est de constater comment ils se généralisent sur d'autres problèmes similaires mais différents.

Le même jeu, V-Ball en l'occurrence, mais comportant un nombre différent de joueurs constitue un problème très similaire, mais qui comporte néanmoins des nuances subtiles. La version à six joueurs est particulièrement intéressante. On dispose dans ce cas là d'un plus grand nombre d'options et certaines de ces options ont un plus grand nombre de variables à prendre en considération dans leurs évaluations. Les réseaux de neurones comptent donc plus d'unités et leur entraînement s'en trouve donc complexifié. Le cas de la partie à seulement trois joueurs permet d'explorer des possibilités d'une très grande complexité, peut-être trop complèxes pour des parties avec un nombre de joueurs plus élevé. On peut ainsi valider des concepts avant de les mettre à l'essai à plus grande échelle. On pourrait par exemple, plutôt que d'avoir une seule option par adversaire pour l'utilisation des habiletés offensives, avoir autant d'options que d'habiletés.

Idéalement, il faudrait tenter de soumettre l'architecture décrite dans cet

ouvrage à l'apprentissage d'un jeu complètement différent. Une grande part du travail serait à refaire, notamment dans le choix des entrées pertinentes, mais l'architecture conçue pour le présent projet peut très bien supporter l'apprentissage d'un autre jeu.

Finalement, une conclusion plus générale que l'on peut tirer des travaux qui ont été réalisés est que l'algorithme de descente de gradient sur la politique est très adapté pour solutionner les processus de décision markoviens qui se déroulent en temps-réel. Il est également possible d'affirmer que pour des problèmes dont les espaces d'états et d'actions sont d'une complexité comparable à notre problème, il n'est pas toujours utile d'employer des couches cachées à l'intérieur des réseaux de neurones.

Bibliographie

[1] Bruce Abramson and Richard E. Korf. A model of two-player evaluation functions. *In Proceedings of the 6th National Conference on Artificial Intelligence*, pages 90–94, 1987.

[2] David H. Ackley and Michael L. Littman. Generalization and scaling in reinforcement learning. *Advances in Neural Information Processing Systems 2*, pages 550–557, 1990.

[3] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison-Wesley, 1988.

[4] John H. Andreae. Stella : A scheme for a learning machine. *Proceedings of the 2nd IFAC Congress*, pages 497–502, 1961.

[5] John H. Andreae. A learning machine with monologue. *International Journal of Man-Machine Studies*, 1 :1–20, 1969.

[6] Marc S. Atkin, David L. Westbrook, and Paul R. Cohen. Capture the flag : Military simulation meets computer games. *Papers from the AAAI Spring Symposium of Artificial Intelligence and Computer Games*, pages 1–5, 1999.

[7] F. Avnaim, J.-D. Boisonnat, and B. Faverjon. A pratical exact motion planning algorithm for polynal objects amidst polygonal obstacles. *Lecture notes in computer Science*, 391 :67–86, 1988.

[8] Jonathan R. Bachrach. A connectionist learning control architecture for navigation. *Advances in Neural Information Processing Systems 3*, pages 457–463, 1991.

[9] Leemon C. Baird. Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base Ohio : Wright Laboratory, 1993.

[10] Leemon C. Baird. Residual algorithms : Reinforcement learning with function approximation. *Proceedings of the Twelth International Conference on Machine Learning*, 12 :30–37, 1995.

[11] Leemon C. Baird and A. H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical report, Wright-Patterson Air Force Base Ohio : Wright Laboratory, 1993.

[12] Karthik Balakrishnan, Rushi Bhatt, and Vasant Honavar. Spatial learning and localization in animals : A computational model and some behavioral experiments. *ECCM-98 : Proceedings of the Second European Conference on Cognitive Modelling*, 1998.

[13] E. Barnard. Temporal difference methods and markov models. *IEEE Transactions on Systems, Man and Cybernetics*, 23 :35–44, 1993.

[14] Andrew G. Barto and C. W. Anderson. Structural learning in connectionist systems. *Program of the Seventh Annual Conference of the Congnitive Science Society*, pages 43–54, 1985.

[15] Andrew G. Barto, C. W. Anderson, and Richard S. Sutton. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 43 :175–185, 1982.

[16] Andrew G. Barto, S. J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72 :81–138, 1995.

[17] Andrew G. Barto and Satinder P. Singh. On the computational economics of reinforcement learning. *Proceedings of the Connectionist Models 1990 Summer School*, pages 35–44, 1990.

[18] Andrew G. Barto and Richard S. Sutton. Goal seeking components for adaptative intelligence : An initial assessment. Technical Report AFWAL-TR-81-1070, Air Force Wright Aeronautical Laboratories/Anionics Laboratory, 1981.

[19] Andrew G. Barto, Richard S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning problems. *IEEE Transactions on systems, Man and cybernetics*, 13 :835–846, 1983.

[20] Andrew G. Barto, Richard S. Sutton, and P. S. Brouwer. Associative search network : A reinforcement learning associative memory. *Biological Cybernetics*, 40 :201–211, 1981.

[21] Andrew G. Barto, Richard S. Sutton, and Christopher J. C. H. Watkins. Sequential decision problems and neural networks. *Advances in Neural Information Processing Systems 2*, pages 686–693, 1990.

[22] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap : A chess program that learns by combining td(lambda) with minimax search. *Technical report, Department of Systems Engineering, Australian National University*, 1997.

[23] Tim Beardsley. Science's survival strategy : Researchers are learning how to live in a new budgetary environment. *Scientific American*, Août 1997. `http://www.scientificamerican.com/0897issue/0897infocus.html`.

[24] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[25] R. E. Bellman. A markov decision process. *Journal of Mathematical Mechanics*, 6 :679–684, 1957.

[26] Dimitri P. Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[27] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 493–498, 1998.

[28] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, London, 1997.

[29] A. Borst, M. Egelhaaf, and H. S. Seung. Two-dimensional motion perception in flies. *Neural Computation*, 5(6), 1993.

[30] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of robotics and automation*, 2(1) :14–23, 1986.

[31] Rodney A. Brooks, Cynthia Breazeal (Ferrell), Robert Irie, Charles C. Kemp, Matthew Marjanovic, Brian Scassellati, and Matthew Williamson. Alternate essences of intelligence. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 961–968, 1998.

[32] Papadimitriou C.H. and Tsitsiklis J.N. The complexity of markov decision processes. *Math. of Operational Research*, 12 :441–450, 1987.

[33] William W. Cohen. Learning from textbook knowledge : A case study. *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990.

[34] William W. Cohen. Abductive explanation-based learning : A solution to the multiple inconsistent explanation problem. *Machine Learning*, 8 :167–219, 1992.

[35] D. Dai and D.T. Lawton. Range-free qualitative navigation. *IEEE Robotics and Automation Conference*, 1993.

[36] Ian Lane Davis. A question of character : Rules to play by. *Papers from the AAAI Spring Symposium of Artificial Intelligence and Computer Games*, pages 18–23, 1999.

[37] Thomas G. Dietterich and Nicholas S. Flann. Explanation-based and reinforcement learning : A unified view. *Machine Learning*, 28(2/3) :169–210, 1997.

[38] Andrew P. Duchon, William H. Warren, and Leslie Pack Kaelbling. Ecological robotics. *Adaptive Behavior : Special Issue on Biologically Inspired Models of Spatial Navigation*, 6(3/4) :473–507, 1998.

[39] Herbert D. Enderton. The golem go program. Technical Report CMU-CS-92-101, School of Computer Science, Carnegie-Mellon University, 1991.

[40] Mark J. Fasciano. Everyday-world plan use. Technical Report TR-96-07, Computer Science Department, University of Chicago, 1996.

[41] Mark J. Fasciano. Real-time case-based reasoning in a complex world. Technical Report TR-96-05, Computer Science Department, University of Chicago, 1996.

[42] Michael R. Genesereth and Nils J. Nelson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufman Publishers, 1987.

[43] Roy Glasius, Andrzej Komoda, and Stan C. A. M. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1) :125–133, 1995.

[44] Eric Goodwin. How a biochemical metabolic model can contribute to intelligent lifelike behaviour. *Papers from the AAAI Spring Symposium of Artificial Intelligence and Computer Games*, pages 51–54, 1999.

[45] S. Grand, D. Cliff, and A. Malhorta. Creatures : Artificial life autonomous software agents for home entertainement. *Proc. of the first International conference on Autonomous Agents*, pages 51–54, 1997.

[46] John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan, 1975.

[47] John Horgan. The deep blue team plots its next move. *Scientific American*, Mars 1996.

[48] Ron Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, 1960.

[49] Feng hsiung Hsu, Thomas Anantharaman, Murray Campbell, and Andreas Nowatzyk. A grandmaster chess machine. *Scientific American*, Octobre 1990.

[50] IBM. Section du site web de ibm consacré aux travaux portant sur le projet deep blue. `http://www.research.ibm.com/deepblue/`.

[51] R. Isaacs. Differential games : A mathematical theory with applications to warefare and other topics. *Technical Report Research Contribution No.1,Center for Naval Analysis*, 1963.

[52] A. E. Bryson Jr. Optimal control-1950 to 1985. *IEEE Control Systems*, 13(3) :26–33, 1996.

[53] G. A. Kimble. *Higard and Marquis' Conditioning and Learning.* Appleton-Century-Crofts, New york, 1961.

[54] Margit Kinder and Wilfried Brauer. Classification of trajectories – extracting invariants with a neural network. *Advances in Neural Information Processing Systems 6*, 1993.

[55] A. H. Klopf. Brain function and adaptative systems - a heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA., 1972.

[56] A. H. Klopf. A comparaison of natural and artificial intelligence. *SIGART Newsletter*, 53 :11–13, 1975.

[57] C. A. Knoblock. Learning abstraction hierarchies for problem solving. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI '90)*, pages 923–928, 1990.

[58] Leonid Kuvayev. Learning to play hearts. *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[59] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991.

[60] Michael Littman. Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[61] William S. Lovejoy. A survey of algorithmic methods for partially observed markov processes. *Annals of Operations Research*, 28 :47–65, 1991.

[62] Richard Maclin and Jude W. Shavlik. Incorporating advice into agents that learn from reinforcements. *Proceedings of the Twelth National Conference on Artificial Intelligence (AAAI '94)*, pages 694–699, 1994. `ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/maclin.aaai94.ps`.

[63] Richard Maclin and Jude W. Shavlik. Combining the predictions of multiple classifiers : Using competitive learning to initialize neural networks. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 524–530, 1995. `ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/maclin.ijcai95.ps`.

[64] Jűrgen Schmidhuber Marco Wiering. Fast online q(lambda). *Machine Learning Journal*, 1998.

[65] J. M. Mendel. A survey of learning control systems. *ISA Transactions*, 5 :297–303, 1966.

[66] Leslie Kaelbling Michael Littman, Anthony Cassandra. Learning policies for partially observable environments : Scaling up. *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, 1995.

[67] Donald Michie. *Trial and error*, pages 129–145. Penguin, Harmondsworth, 1961.

[68] Donald Michie and R. A. Chambers. *BOXES : Sn experiment in adaptative control*, pages 137–152. Oliver and Boyd, Edinburgh, 1968.

[69] Marvin L. Minsky. *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain Model Problem.* PhD thesis, Princeton University, 1954.

[70] Marvin L. Minsky. Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49 :8–30, 1961.

[71] David Jack Mostow. Machine transformation of advice into a heuristic search procedure. *Machine Learning : An Artificial Intelligence Approach*, pages 367–403, 1983.

[72] Schraudolph N. N., Dayan Peter, and Sejnowski T. J. Using the td(λ) algorithm to learn an evaluation function for the game of go. *Advances in Neural Information Processing Systems*, 6, 1994.

[73] Yun Seok Nam and Bum Hee Lee. An analytic approach to moving obstacle avoidance using an artificial potential field. *Proceedings of the International Conference on Intelligent Robots and Systems (IROS '95)*, 1995.

[74] Daniel Kenneth Olson. Learning to play games from experience : An application of artificial neural networks and temporal difference learning. Master's thesis, Pacific Lutheran University, 1993.

[75] P.I. Pavlov. *Conditionned Reflexes.* Oxford University Press, London, 1927.

[76] M.L. Putterman. *Markov Decision Problems.* Wiley, New York, 1994.

[77] Bjorn Reese and Bryan Stout. Finding a pathfinder. *Papers from the AAAI Spring Symposium of Artificial Intelligence and Computer Games*, pages 69–72, 1999.

[78] F. Rosenblatt. *Principles of Neurodynamics : Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington, D.C., 1962.

[79] G.A. Rummery and M.Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR, Cambridge University, 1994.

[80] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on research and development*, 3 :211–229, 1959.

[81] D. G. Schultz, Peter Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275 :1593–1598, 1997.

[82] C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41 :256–275, 1950.

[83] John W. Sheppard and Steven L. Salzberg. Memory based learning of pursuit games. Technical Report JHU-93/94-02, Johns Hopkins University, 1994.

[84] John W. Sheppard and Steven L. Salzberg. Combining genetic algorithms and memory based reasoning in pursuit games. *Proceedings of the 1995 International Conference on Genetic Algorithms*, 1995.

[85] Sattinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility trace. *Machine Learning*, 22 :123–158, 1996.

[86] Paul Smaglik. Funding mechanisms affect research culture. *The Scientist*, 13(20), Octobre 1999. `http://www.the-scientist.library.upenn.edu/yr1999/oct/prof_991011.html`.

[87] E.J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 19 :1300–1322, 1971.

[88] Antony Stentz. The focussed d* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Inteligence*, 1995. `http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.ps`.

[89] W. Bryan Stout. Smart moves : Intelligent path-finding. *Game Developper Magazine*, Octobre 1996. `http://www.gamasutra.com/features/programming/19990212/sm_01.htm`.

[90] Richard S. Sutton. Single chanel theory : A neuronal theory of learning. *Brain Theory Newsletter*, 4 :72–75, 1978.

[91] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3 :9–44, 1988.

[92] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning.* The MIT Press, 1998.

[93] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Andvances in Neural Information Processing Systems*, 12, 1999.

[94] Richard S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps : Learning, planning, and representing knowledge at multiple temporal scales. *Journal of Artificial Intelligence Research*, 1 :1–39, 1998.

[95] G. J. Tesauro. Td-gammon, a self-teaching backgammon program, acheives master-level play. *Neural Computation*, 6 :215–219, 1994.

[96] G. J. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38 :58–68, 1995.

[97] Sandip Sen Thomas Hayes. Evolving behavioral strategies in predator and prey. *Proceedings of the 1995 IJCAI Workshop*, 1995.

[98] E.L. Thorndike. *Animal Intelligence.* Hafner,Darien,CT, 1911.

[99] Sebastian Thurn and Anton Schwartz. Issues in using function approximation for reinforcement learning. *Proc. of the 1993 Connectionist Summer School*, pages 255–263, 1993.

[100] Walter F. Tichy. Should computer scientists experiment more ? *IEEE Computer*, pages 32–40, Mai 1998.

[101] Michael I. Jordan Tommi Jaakkola, Satinder P. Singh. Reinforcement learning algorithm for partially observable markov decision problems. *Andvances in Neural Information Processing Systems 7*, pages 345–352, 1995.

[102] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59 :433–460, 1950.

[103] A. Frank van der Stappen and Mark H. Overmars. Motion planning amidst fat obstacles. *Proceedings of the tenth annual symposium on Computational Geometry*, pages 31–40, 1994.

[104] M. D. Waltz and K. S. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10 :390–398, 1965.

[105] Christopher J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989.

[106] Christopher J. C. H. Watkins and Peter Dayan. Technical note : Q learning. *Machine Learning*, 8 :279–292, 1992.

[107] B. Widrow, N. K. Gupta, and S. Maitra. Punish/reward : Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 3 :455–465, 1973.

[108] Nathaniel Scott Winstead. Some explorations in reinforcement learning techniques aplied to the problem of learning to play pinball. *Papers from AAAI Workshop on Entertainment and AI/A-Life*, pages 1–5, 1996.

[109] Ian H. Witten. An adaptative optimal controller for discrete-time markov environments. *Information and Control*, 34 :286–295, 1977.

[110] Steven Woodcock. Game ai : State of the industry. *Game Developper Magazine*, Octobre 1998. `http://www.gamasutra.com/features/programming/19981120/gameai_01.htm`.

[111] Steven Woodcock. Game ai : State of the industry. *Game Developper Magazine*, Août 1999. `http://www.gamasutra.com/features/19990820/game_ai_01.htm`.