



Le transfert adaptatif en apprentissage par renforcement : application à la simulation de schéma de jeux tactiques

Aydano Pamponet Machado

► To cite this version:

Aydano Pamponet Machado. Le transfert adaptatif en apprentissage par renforcement : application à la simulation de schéma de jeux tactiques. Modeling and Simulation. Université Pierre et Marie Curie - Paris VI, 2009. French. <NNT : 2009PA066209>. <tel-00814207>

HAL Id: tel-00814207

<https://tel.archives-ouvertes.fr/tel-00814207>

Submitted on 16 Apr 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE DE DOCTORAT DE L'UNIVERSITE PARIS 6
PIERRE ET MARIE CURIE

Spécialité

INFORMATIQUE

Présentée par

AYDANO PAMPONET MACHADO

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE PARIS 6

Sujet de la thèse

**Le transfert adaptatif en apprentissage par renforcement
Application à la simulation de schéma de jeux tactiques**

Soutenue le 24 juin 2009

Devant le jury composé de :

Yann Chevaleyre	Encadrant	Maître de Conférences à l'Université Paris-Dauphine
Amal El Fallah Seghrouchni	Examinateur	Professeur à l'Université Pierre et Marie Curie
Frédéric Garcia	Rapporteur	Directeur de Recherches à l'INRA
Philippe Preux	Rapporteur	Professeur à l'Université de Lille 3
Geber Ramalho	Examinateur	Professeur à l'Universidade Federal de Pernambuco
François Rioult	Invité	Maître de Conférences à l'Université de Caen
Hubert Ripoll	Invité	Professeur à l'Université de la Méditerranée
Jean-Daniel Zucker	Directeur	Directeur de Recherches à l'IRD

*A ma petite famille : mon épouse Janiffer et mon fils Loïc.
Je vous aime de tout mon cœur.*

« Learning is not attained by chance, it must be sought for with ardor and attended to with diligence.. »

Abigail Adams (1744 - 1818), 1780.

Remerciements

Je voudrais commencer en exprimant tout ma reconnaissance à mon encadrant de thèse, Yann Chevaleyre, Maître de Conférences à l’Université Paris-Dauphine. Tout le travail que j’ai effectué durant ces années de thèse doit beaucoup à son soutien qu’il m’a toujours confié, aussi bien sur le plan scientifique que sur le plan moral. Son enthousiasme et son assistance ont été fondamentaux pendant cette thèse et pour ma formation au métier de chercheur en informatique.

Je remercie infiniment mon directeur de thèse Jean-Daniel Zucker, Directeur de Recherches à l’Institut de Recherche pour le Développement, pour toute la confiance, le support et l’encadrement qu’il m’a apporté. Ses conseils et ses commentaires ont été forts utiles dans le cadre de cette thèse.

Je tiens à remercier Frédéric Garcia, Directeur de Recherches à l’IRNA, et Philippe Preux, Professeur à l’Université de Lille 3, d’être intéressés à mes travaux et d’avoir accepté d’être rapporteur ma thèse. Je suis très reconnaissant pour le temps que vous m’avez accordé.

Je remercie tout particulièrement à Geber Ramalho, Professeur à l’Universidade Federal de Pernambuco, pour toute sa confiance et son support pendant tous ces années. Ses conseils ont été très importants pour ma formation professionnelle et individuelle. Merci pour toutes ces opportunités et d’avoir accepté d’être un des examinateurs de mon travail de thèse.

Je tiens aussi à remercier Amal El Fallah Seghrouchni, Professeur à l’Université Pierre et Marie Curie, Hubert Ripoll, Professeur à l’Université de la Méditerranée, François Rioult, Maître de Conférences à l’Université de Caen Basse-Normandie, qui on bien voulu faire partie de mon jury de thèse.

Je remercie Mohamed SEBBANE, entraîneur et docteur au LSIS à Marseille, Hubert Ripoll, Professeur à l’Université de la Méditerranée, et tout le groupe du LSIS pour tout le support professionnel dans le domaine de STAPS et du football.

Je remercie également, David Elkaïm, qui a fait un lourd travail de relecture sans compter son effort d'ajouter un cours de français en même temps , Julien Bourdaillet et Benoît Calvez qui ont également contribués à la qualité de ce document.

Il a été un plaisir de faire partie de l'équipe ACASA, de l'équipe SMA et du LIP6. Et je remercie chaleureusement tous ceux avec qui j'ai pu partager mes journées ou que j'ai rencontré au LIP6. Un merci tout spécial à mes amis Julien Velcin, Julien Bourdaillet, David Elkaïm et évidemment à mes amis et compatriotes Charles Madeira et Giordano Cabral qui ont ajouté les couleurs du Brésil aux couloirs du LIP6.

Je remercie tout le support, gentillesse et compétence de Ghislaine Mary et Thierry Lanfroy qui m'ont aidé à sortir plusieurs fois du dédale administratif. J'en profite pour remercier Christophe Bouder et Jean-Pierre Arranz pour le support technique au niveau d'environnement de travail et à la CAPES pour le support financier.

Je tiens à remercier chaleureusement les familles Elkaïm, Peyre et Madeira pour nous avoir adoptés, en nous faisant sentir un peu plus chez nous avec tous les moments que nous passons ensemble.

Pour finir, je remercie tous les membres de ma famille qui m'ont toujours témoigné une énorme confiance. Je remercie de tout mon cœur mes parents (Robério et Ana Machado) et mon frère (Alison) pour tout ce qu'ils ont fait pour que je puisse arriver à ce moment. Je remercie ma femme Janiffer et mon fils Loïc pour la confiance et le soutien sans lesquels rien n'aurait été possible.

Résumé

L'un des principaux objectifs de l'apprentissage par renforcement est de développer des algorithmes capables de générer des politiques de bonne qualité en un temps le plus réduit possible. Les progrès dans ce domaine sont tels que les performances de certains algorithmes récents approchent des limites théoriques. Malheureusement, la plupart des tâches d'apprentissage issues du monde réel sont de grande dimension, et l'apprentissage prend dès lors un temps considérable.

Pour accélérer l'apprentissage, l'une des voies possibles consiste à guider le processus d'exploration à l'aide de connaissances du domaine. Lorsque ces connaissances prennent la forme d'une politique apprise précédemment sur une tâche reliée à la tâche courante, on parle de transfert de politique. La plupart des algorithmes de transfert de politique existants sont basés sur une hypothèse implicite : ils supposent que la politique disponible est d'une bonne qualité sur la tâche courante. Clairement, lorsque cette hypothèse n'est pas respectée, les performances de ces algorithmes se dégradent bien en dessous des performances des méthodes d'apprentissage par renforcement standards.

Le but de cette thèse est de lever cette hypothèse, en proposant des algorithmes de transfert de politique capables de s'adapter à la qualité de la politique disponible. Plus précisément, nous introduisons un paramètre nommé le taux de transfert, qui contrôle à quel point l'algorithme se fiera à la politique disponible. De plus, nous proposons d'optimiser ce taux afin de faire le meilleur usage de cette politique. Ainsi, les algorithmes que nous proposons dans cette thèse offrent une certaine robustesse face à la politique disponible, ce qui n'était pas le cas des approches précédentes.

Ces algorithmes sont évalués sur deux domaines différents : un problème jouet (le gridworld), et une application d'aide à l'entraîneur de football. Cette dernière application propose à un entraîneur de saisir des schémas tactiques à l'aide d'une interface graphique, et lui permet ensuite de visualiser des agents-joueurs en train de réaliser ces mêmes schémas. Pour satisfaire dans des délais raisonnables la requête de l'entraîneur, l'apprentissage par renforcement seul ne suffit pas, et nos algorithmes de transfert ont été appliqués sur ce domaine avec succès.

Mots clés : transfert de connaissance, apprentissage par renforcement, systèmes multiagent, gridworld, simulation, situation de jeu

Abstract

A possible way to accelerate reinforcement learning process is to guide the exploration process using prior domain knowledge. This called knowledge transfer, and most transfer algorithms are based on an implicit assumption. They suppose that prior knowledge has a good quality for the current task. If this condition is not true, the learning process will be worse than standard reinforcement learning algorithm (negative transfer).

This thesis put forwards some transfer algorithms to avoid this problem, whose can adapts learning process to prior knowledge quality. More precisely, we introduce a parameter called transfer rate, which controls how much prior knowledge will be used. In addition, we propose to optimize the transfer rate in order to make the best use of this policy. Thus, the proposed algorithms provide some robustness, working for all prior knowledge quality level, which was not the case with previous approaches.

These algorithms are evaluated in two different problems: a toy problem (the gridworld), and a real complex one (a coach assistant tool). The latter application offers a coach to seize tactical patterns with a graphical interface, and then allows agents to view players doing the same patterns. To meet within a reasonable time, the request of the coach. The reinforcement learning alone is not enough, and transfer our algorithms have been applied to this area with success.

Keywords: knowledge transfer, reinforcement learning, multiagent systems, gridworld, simulation, game situation

Table des matières

REMERCIEMENTS	7
RESUME	9
ABSTRACT	11
TABLE DES MATIERES	13
PREAMBULE	17
PARTIE I :	
APPRENTISSAGE PAR RENFORCEMENT ET TRANSFERT ADAPTATIF.....	21
CHAPITRE 2	
APPRENTISSAGE PAR RENFORCEMENT	23
2.1 ELEMENTS D'APPRENTISSAGE PAR RENFORCEMENT	24
2.2 LES PROCESSUS DE DECISION MARKOVIENS	25
2.2.1 La Politique.....	27
2.2.2 Les mesures de gain	27
2.2.3 La Fonction d'utilité	29
2.2.4 Les Politiques optimales	30
2.3 EXPLORATION ET EXPLOITATION	30
2.3.1.1 La stratégie ϵ -gloutonne.....	31
2.3.1.2 La stratégie boltzmannienne	32
2.4 LES PRINCIPALES METHODES	32
2.4.1 Programmation dynamique	32
2.4.2 Les méthodes de Monte Carlo	33
2.4.3 Les méthodes de différences temporelles.....	35
2.4.3.1 <i>Sarsa</i>	36
2.4.3.2 <i>Q-learning</i>	37
2.5 L'UNIFICATION DE MONTE-CARLO ET DES METHODES DE DIFFERENCES TEMPORELLES	38
2.6 APPRENTISSAGE DANS DES ESPACES DE GRANDE TAILLE	39
2.6.1 Généralisation par approximation de fonctions	40
2.6.1.1 Cas général.....	41
2.6.1.2 Cas linéaire	42
Codage en carreaux (<i>Tile Coding</i>).....	42
2.7 CONCLUSION	44
CHAPITRE 3	
ETAT DE L'ART DU TRANSFERT DE CONNAISSANCE.....	45
3.1 BASES DU TRANSFERT DE CONNAISSANCE EN APPRENTISSAGE PAR RENFORCEMENT.....	45
3.2 PROBLEMATIQUES DU TRANSFERT EN APPRENTISSAGE PAR RENFORCEMENT.....	48
3.3 METHODES ACTUELLES DE TRANSFERT	49
3.3.1 Apprendre en observant un autre agent.....	50
3.3.2 Transfert des valeurs (ou des <i>Q</i> -valeurs).....	51
3.3.3 Réutilisation de politique	53

3.3.4 Les récompenses façonnées et les estimateurs de progrès	55
CHAPITRE 4	
OPTIMISATION DU TRANSFERT DE CONNAISSANCE EN APPRENTISSAGE PAR	
RENFORCEMENT.....	59
4.1 PLATEFORME D'ESSAI ET D'EXPERIMENTATION.....	60
4.2 LES POLITIQUES <i>K</i> -ALTEREES.....	61
4.3 LIMITE DU TRANSFERT DES POLITIQUES <i>K</i> -ALTEREES.....	63
4.4 LA NOTION DE TAUX DE TRANSFERT	63
4.5 INFLUENCE DU TAUX DE TRANSFERT SUR L'APPRENTISSAGE	65
4.6 L'ALGORITHME DE TRANSFERT ADAPTATIF	67
4.7 ÉVALUATION HORS LIGNE.....	69
4.8 IMPLEMENTATION DE LA FONCTION <i>OPTIMISE_φ(π̄, π,...)</i>	69
4.9 ÉVALUATION EN LIGNE.....	71
4.10 EXPERIMENTATIONS ET RESULTATS.....	72
4.10.1 Procédure d'expérimentation avec le <i>gridworld</i>	73
4.10.2 Résultats avec le <i>gridworld</i>	78
4.11 DISCUSSIONS	91
4.12 CONCLUSIONS.....	92
PARTIE II :	
APPLICATION AUX LOGICIELS D'ENTRAINEMENT SPORTIF ASSISTÉ PAR ORDINATEUR....	93
CHAPITRE 5	
ETAT DE L'ART DE L'ENTRAINEMENT SPORTIF ASSISTÉ PAR ORDINATEUR	95
5.1 MOTIVATION ET DEFINITION DU PROBLEME AVEC L'APPROCHE ENVISAGEE	96
5.2 INTRODUCTION AUX SYSTEMES D'AIDE A L'ENTRAINEUR	97
5.2.1 Le rôle de l'entraîneur et les schémas tactiques.....	97
5.2.2 Les logiciels d'aide à l'entraîneur	98
5.2.3 Limitation des logiciels d'aide à l'entraîneur	100
5.3 LA SIMULATION SPORTIVE D'AUJOURD'HUI.....	102
5.3.1 Les jeux vidéo de simulation sportive.....	103
5.3.2 La simulation d'équipes sportives, l'exemple de la <i>RoboCup</i>	107
5.3.3 La simulation sportive d'aujourd'hui et l'entraînement sportif	108
CHAPITRE 6	
CONCEPTION D'UNE EQUIPE D'AGENTS APPRENANTS POUR LA SIMULATION DE	
SITUATIONS DE JEU	111
6.1 UNE APPROCHE POUR LE DEPLOIEMENT AUTONOME ET ADAPTABLE DE SITUATIONS DE JEUX	112
6.2 INTERACTION AVEC L'UTILISATEUR.....	113
6.3 CONCEPTION DES AGENTS.....	114
6.3.1 Déplacement et actions	115
6.3.1.1 L'espace d'actions.....	115
6.3.2 Sélection des actions	116
6.4 RAISONNEMENT ET LA SELECTION D'ACTIONS.....	116
6.4.1 Apprendre par renforcement	117
6.4.1.1 Représentation de l'espace d'états	118
6.4.1.2 Récompenses.....	119
6.4.1.3 Algorithme d'apprentissage et transfert de connaissance	119
6.5 QUEL TYPE D'APPRENTISSAGE MULTIAGENT	121
6.6 EXPERIMENTATIONS ET RESULTATS.....	122
6.6.1 Configuration des expérimentations	123
6.6.2 Résultats.....	125
6.7 CONCLUSIONS	132
CHAPITRE 7	
CONCLUSIONS ET PERSPECTIVES	133
BIBLIOGRAPHIE	137
ANNEXE A	
ALGORITHME POUR PERTURBER UNE POLITIQUE	147

ANNEXE B	
STEERING BEHAVIORS	149
ANNEXE C	
DETAILS D'IMPLEMENTATION DU SYSTEME.....	159
ANNEXE D	
CORRESPONDANCE ENTRE LES DIAGRAMMES ET LES DESCRIPTIONS UTILISEES.....	167
ANNEXE E	
STRUCTURE DE DONNEES DU SYSTEME.....	177
ANNEXE F	
SENSIBILITE DU TRANSFERT DES VALEURS A LA QUALITE DE LA POLITIQUE DE TRANSFERT.....	189

Chapitre 1

Préambule

Une des difficultés de l'entraîneur d'une équipe est de se faire comprendre par tous les membres de son groupe. Afin de réduire cette difficulté, les entraîneurs font appels à des outils informatiques pour faciliter la communication. Cependant nous avons appris, grâce à une coopération avec le Laboratoire des Sciences de l'Information et des Systèmes (LSIS)¹, que ces outils ne sont pas satisfaisants pour les professionnels du sport.

En discutant avec le groupe de chercheurs en STAPS (Sciences et Techniques des Activités Physiques et Sportives) du LSIS, nous avons identifié les caractéristiques de l'outil idéal pour les entraîneurs. Dans ce cas, nous avons envisagé un outil de simulation de situations de jeu où l'entraîneur disposerait d'agents avec des caractéristiques proches des membres de son équipe pour réaliser la situation de jeu, et où il pourrait interagir avec les agents à sa manière. Comme par exemple, en faisant la description de la situation de jeu de la même façon que sur un tableau noir (Figure 1.1).

¹ <http://www.lsis.org/>

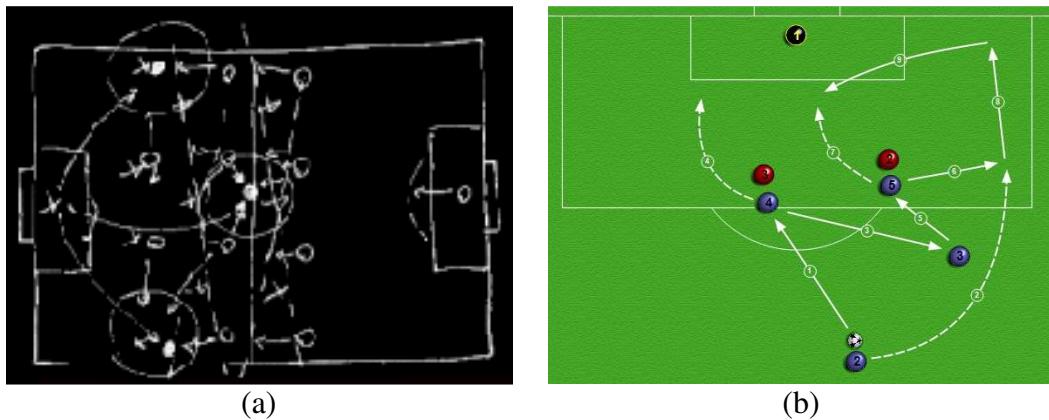


Figure 1.1 – Exemple de la description graphique de situations de jeu sur un tableau noir : (a) version classique ; (b) version numérique.

Doter ces agents de caractéristiques des vrais joueurs ne revient pas seulement à les programmer pour une situation spécifique mais à leur donner la capacité de s'adapter à de nouvelles situations de jeu, ou au changement de la situation actuelle, ou encore de s'améliorer avec l'expérience.

Dans le but d'avoir un outil conforme aux besoins des professionnels du sport, nous avons envisagé une approche basée sur des techniques d'Intelligence Artificielle pour concevoir des agents autonomes et plus proches des vrais joueurs. L'étude et la conception de cette approche ont été la motivation principale pour la réalisation de cette thèse qui apporte sa contribution à la simulation de situations de jeu et au domaine du transfert de connaissance pour l'apprentissage par renforcement.

Les chercheurs en Intelligence Artificielle (IA), ou plus précisément en Apprentissage par Renforcement, étudient comment programmer des ordinateurs ou des robots capables d'apprendre à partir de leurs expériences.

Cette activité a déjà produit de meilleurs résultats que les solutions utilisées auparavant pour des problèmes réels et complexes, notamment : le meilleur joueur de backgammon du monde (Tesauro, 1990; Tesauro, 1992; Tesauro, 1995; Tesauro, 2002) (Dahl, 1998- 2004) ; le perfectionnement du contrôleur d'ascenseur (Crites et Barto, 1996; Crites et Barto, 1998) ; un contrôle de stock 10-15% meilleur que les méthodes standard de l'industrie (Van Roy *et al.*, 1997) ; la meilleure méthode d'allocation dynamique de canaux radio pour la téléphonie mobile (Singh et Bertsekas, 1997) (Nie et Haykin, 1999; Nie et Haykin, 1999) etc.

Des exemples comme ces derniers montrent que les algorithmes d'apprentissage par renforcement sont capables de résoudre des problèmes complexes. Toutefois, lorsque les agents commencent leur processus d'apprentissage sans aucune connaissance (*agents tabula*

rasa), la maîtrise de tâches difficiles devient souvent très lente ou impossible. Et donc, une quantité importante des travaux de recherche en apprentissage par renforcement se concentrent sur l'augmentation de la vitesse d'apprentissage en exploitant de différentes manières la connaissance d'un expert du domaine.

L'utilisation d'abstractions sur l'espace d'états (p. ex.: la généralisation par approximation de fonctions (Sutton et Barto, 1998), la décomposition hiérarchique d'une tâche en plusieurs sous-tâches (Dietterich, 1998) et l'utilisation d'abstractions temporelles pour une séquence d'actions à la place de travailler avec les actions atomiques (ou à une étape) (Sutton *et al.*, 1999) sont des exemples d'approches couramment utilisées pour accélérer la vitesse d'apprentissage. Cependant, l'expertise utilisée pour mettre au point l'algorithme d'apprentissage va aider l'agent à apprendre plus vite, mais il va toujours réaliser son exploration en choisissant des actions au hasard et cela ne vas pas éviter l'exploration d'une mauvaise région du problème.

Une mauvaise stratégie d'exploration est toujours un inconvénient pour les problèmes avec un grand espace d'états et actions. Cela peut beaucoup ralentir le processus d'apprentissage. Ainsi, la connaissance d'un expert du domaine, ou une éventuel connaissance disponible, pourrait être utilisée pour créer une heuristique d'exploration au lieu d'être seulement utilisé pour la mise au point de l'algorithme d'apprentissage.

L'utilisation de l'information disponible pour enrichir la connaissance de l'agent ou pour l'aider à choisir ses actions, en guidant l'exploration, est ce que nous dénommons transfert de connaissance.

L'approche que nous avons choisie pour le problème de la simulation de situations de jeu ajoute cette capacité au processus d'apprentissage. Car nous avons à notre disposition un système à base de règles qui décrit d'une façon simple le comportement fondamental d'un joueur. Le problème ce que le système à base de règles ne marche pas pour toutes les situations de jeu. Afin d'éviter le transfert négatif et de bien utiliser la connaissance disponible, nous avons créé un algorithme de transfert adaptatif qui est robuste et général étant donné qu'il s'adapte à la qualité de cette connaissance de transfert disponible et qu'il est indépendant de la façon que cette connaissance est fournie.

Dans un premier temps, notre algorithme est étudié, développé et expérimenté sur le problème du *gridworld* qui est bien connu par les chercheurs du domaine de l'IA. Ce problème est bien plus simple que le problème de la simulation de jeu et cela nous a permis de ne nous concentrer que sur le problème du transfert. En utilisant toujours le *gridworld*, notre

algorithme a été comparé avec les principaux travaux de transfert de connaissance en apprentissage par renforcement.

Dans un deuxième temps, notre algorithme a été mis à l'épreuve dans le problème de la simulation de situations de jeu. Ce dernier est un problème concret et complexe pour l'apprentissage par renforcement et englobe une problématique beaucoup plus large.

Le présent manuscrit de thèse peut être divisé en deux parties :

1. Dans la première partie nous présentons l'apprentissage par renforcement et transfert adaptatif. Cette partie est composée d'une première partie plus théorique, intégrant les bases de l'apprentissage par renforcement (Chapitre 2) et un état de l'art du transfert de connaissance en apprentissage par renforcement (Chapitre 3). Ensuite, nous décrivons dans le Chapitre 4 notre algorithme de transfert de connaissance ainsi que son expérimentation et comparaison avec les algorithmes existants en utilisant le *gridworld* ;
2. La deuxième partie est une application aux logiciels d'entraînement sportif assisté par ordinateur. Cette partie valide notre approche et notre algorithme sur un problème concret : la conception d'une équipe d'agents apprenants pour la simulation de situations de jeu (Chapitre 6). Cette partie est également composée d'une description de l'état de l'art du domaine (Chapitre 5).

Partie I :

**Apprentissage par renforcement et
transfert adaptatif.**

Chapitre 2

Apprentissage par renforcement

Ce chapitre présente les principes généraux de l'apprentissage par renforcement. Il ne se veut pas exhaustif, mais permettra au lecteur d'avoir une vision de l'apprentissage par renforcement. Les concepts et définitions présentés dans ce chapitre aideront à une bonne compréhension de notre travail de recherche et également à la présentation de la bibliographie (Chapitre 3). Pour la même raison, ce chapitre est limité aux approches non basées sur un modèle. Et pour le réaliser, nous nous sommes inspirés de : (Sutton et Barto, 1998), (Bertsekas et Tsitsiklis, 1996), (Kaelbling *et al.*, 1996), (Cornuéjols et Miclet, 2002) et (Russell et Norvig, 1995).

Le but de l'apprentissage par renforcement est d'apprendre ce qu'il convient de faire en différentes situations, de façon à optimiser une récompense numérique au cours du temps. Cet apprentissage s'effectue par interaction avec l'environnement au moyen d'un processus d'essais et erreur. On considère ce type d'apprentissage comme la construction d'une fonction, appelée *stratégie* ou *politique*, qui associe à l'état actuel la prochaine action à exécuter de manière à obtenir la meilleure récompense à long terme.

Pour certains problèmes, une action exécutée n'affecte pas seulement le prochain état et la récompense immédiate, mais également les états subséquents et, par conséquent, toutes les récompenses suivantes. Autrement dit, il y a des cas dans lesquels il faut attendre un certain temps pour voir le résultat des actions qui sont exécutées (par exemple, c'est seulement à la fin d'une partie d'échecs que l'on dispose de la sanction : perte, gain ou nulle). C'est ce qu'on appelle *renforcement retardé*.

Le processus d'essais et d'erreurs, qui permet de trouver l'action optimale à effectuer pour chacune des situations, et *la récompense retardée*, qui permet d'apprendre à sacrifier une récompense à court terme pour obtenir une plus forte récompense à long terme, sont les deux caractéristiques les plus importantes de l'apprentissage par renforcement. (Sutton et Barto, 1998)

2.1 Eléments d'apprentissage par renforcement

Au-delà de l'agent et de l'environnement, on peut identifier quatre composantes principales intervenant dans l'apprentissage par renforcement (Sutton et Barto, 1998) : la *politique* (stratégie), la *fonction de renforcement*, la *fonction d'utilité* et, optionnellement, le *modèle de l'environnement*.

La politique définit la façon dont l'agent apprenant se comporte à un instant donné. Une politique associe des actions à effectuer aux états observés de l'environnement. Ainsi quand l'agent se trouve dans ces états il effectue ces actions. Elle correspond à ce qui en psychologie s'appelle un ensemble de règles de stimulus-réponse ou réflexes conditionnés. Dans certains cas, la politique peut être une simple fonction ou une table de correspondance, mais elle peut également impliquer un calcul complexe comme un processus de recherche. La politique est le cœur de l'agent qui apprend par renforcement : elle suffit pour déterminer son comportement.

La fonction de renforcement définit l'objectif de l'apprentissage par renforcement. Son principe est de faire correspondre à chaque état (ou état-action) de l'environnement une récompense (sous forme scalaire) indiquant son intérêt. L'objectif de l'agent apprenant dans un problème d'apprentissage par renforcement est de maximiser la somme des récompenses perçues. La fonction de renforcement décrit si l'événement est bon ou mauvais pour l'agent. Dans un système biologique, une métaphore des récompenses pourrait être le plaisir et la douleur.

La fonction d'utilité est une estimation du gain que l'on peut espérer depuis un état particulier. Alors que la fonction de renforcement est un indicateur immédiat de la qualité d'une action, la fonction d'utilité est un indicateur de la qualité du choix au long terme. Autrement dit, la récompense associée à un état est un indicateur de gain immédiat, alors que l'utilité d'un état est un estimateur des gains à venir. Par exemple, un état peut ne rapporter qu'une petite récompense dans l'immédiat, mais avoir une grande utilité s'il est régulièrement suivi d'autres états qui rapportent des récompenses élevées.

Enfin, *le modèle de l'environnement* est une composante optionnelle utilisée par certains systèmes d'apprentissage par renforcement. Il s'agit d'un modèle qui simule le comportement de l'environnement. Etant donnés un état et une action, le modèle prédit le prochain état et la prochaine action. L'apprenant peut alors chercher à résoudre explicitement un problème de planification sur son modèle de l'environnement. Il existe également des systèmes qui apprennent par essais et erreurs le modèle de l'environnement, et qui en même temps utilisent ce dernier pour réaliser des planifications.

2.2 Les Processus de Décision Markoviens

Dans cette section, nous allons formaliser les concepts introduits par la théorie actuelle de l'apprentissage par renforcement. En particulier, nous introduisons le cadre formel standard : les Processus de Décision Markoviens (PDM) totalement observables. Il existe un grand nombre d'autres modèles, dont les Processus de Décision Markovien Partiellement Observables (PDMPO) (Kaelbling *et al.*, 1998; Sondik, 1971), les PDM Multiagent (PDMM) (Boutilier, 1999), etc. Cependant on se restreint aux PDM car leur complexité algorithmique est faible et dans l'application qui a motivé ce travail, on est plus proche de l'observabilité totale que partielle.

Dans un PDM, l'agent interagit avec l'environnement par pas de temps discrets, $t = 0, 1, 2, 3, \dots$ à chaque pas de temps t de l'algorithme, l'agent reçoit l'état de l'environnement $s_t \in S$. Il choisit une action $a_t \in A(s_t)$ et reçoit de l'environnement un nouvel état s_{t+1} et une récompense $r_{t+1} \in R$. (Figure 2.1)

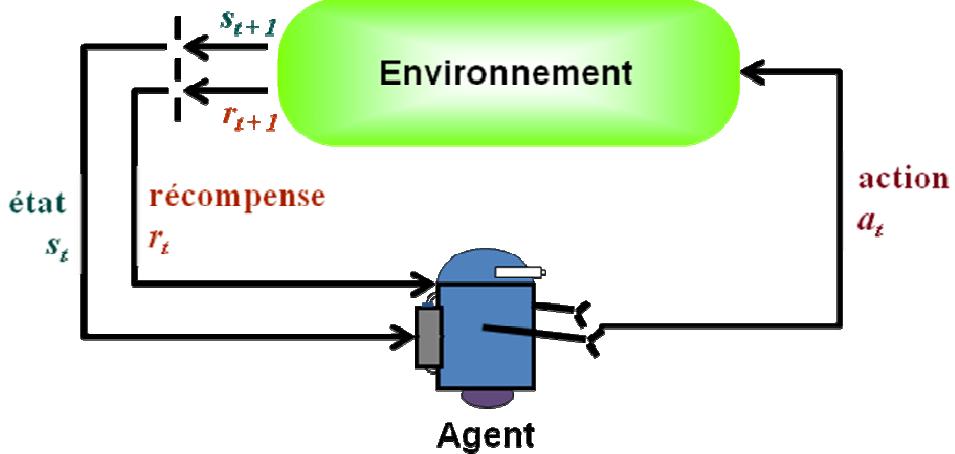


Figure 2.1 - Interaction de l'agent avec l'environnement (Sutton et Barto, 1998)

L'évolution de l'environnement est régie par le Processus de Décision Markovien (PDM) qui décrit l'évolution de l'état et de la récompense en fonction des actions de l'agent.

Formellement, un PDM est défini comme une tuple $\langle S, A, T, R \rangle$ où :

- S est l'ensemble fini d'états ;
- A est l'ensemble fini d'actions ;
- $T : S \times A \times S \rightarrow [0;1]$ est la fonction de transition du système, où $T(s, a, s') = P\{s_{t+1} = s' | s_t = s, a_t = a\}$ indique la probabilité de transition d'un état s à l'instant t vers un état s' à l'instant $t+1$, grâce à l'action a ;
- $R : S \times A \times S \rightarrow \mathbb{R}$ est la fonction récompense, où $R(s, a, s') = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ est l'espérance mathématique de la fonction de renforcement (récompense ou punition) pour la transition d'un état s vers un état s' .

Les deux fonctions R et T sont souvent stochastiques et vérifient la propriété de Markov. La propriété de Markov spécifie que le prochain état de l'environnement ne dépend que de l'état actuel et de l'action qui vient d'être exécutée. Elle se traduit ainsi : $\forall s \in S P\{s_{t+1} = s' | s_t, a_t\} = P\{s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0\}$, ce qui garantit que la fonction de transition est indépendante des interactions passées.

D'autres éléments importants de l'apprentissage par renforcement dans les PDM sont présentés dans les sections suivantes.

2.2.1 La Politique

À chaque étape suite à sa perception s , l'agent sélectionne une action a . On appelle *politique de Markov déterministe* π une fonction $S \rightarrow A$ qui associe à chaque perception s , une action $\pi(s)$ à effectuer. Cette politique détermine donc le comportement de l'agent.

D'une manière générale, on utilise des *politiques de Markov non déterministes*, qui associent à chaque état s et à chaque action a , la probabilité $\pi(s, a)$ que l'agent choisisse l'action a suite à la perception s . La politique est donc définie par $\pi : S \times A \rightarrow [0;1]$ (avec $\sum_a \pi(s, a) = 1$).

Il existe des politiques non markoviennes, c.-à-d., qui dépend de tout l'historique du vécu de l'agent. Cependant, dans un PDM, la politique optimale est markovienne et déterministe.

Le but de l'apprentissage par renforcement est de trouver la politique optimale π^* maximisant la récompense à long terme.

2.2.2 Les mesures de gain

Il faut préciser qu'il n'y a pas de mesure de gain universelle valable pour toutes les situations. Chaque domaine d'application est susceptible d'avoir sa mesure adaptée.

Prenons par exemple un joueur d'échec. Même s'il est sensible au nombre de pièces gagnées ou perdues en cours de parties, à certains critères tels que le contrôle du centre, le plus intéressant pour lui est sans aucun doute le résultat final de la partie : gain, perte ou nulle.

De même, un agent logiciel trader, programmé pour vendre et acheter des actions dans une salle de marché, pourra être programmé pour privilégier les gains à court ou à long terme, selon les préférences de son propriétaire.

En général, on s'intéresse à une mesure de gain cumulée dans le temps. Il existe plusieurs manières de le faire, cependant il y a trois mesures ont été plus particulièrement distinguées dans les travaux de recherches sur l'apprentissage par renforcement :

- Gain cumulé avec horizon fini :

$$R = r_0 + r_1 + r_2 + r_3 + \dots + r_T = \sum_{k=0}^T r_k \quad (2.1)$$

- Gain cumulé avec intérêt et horizon infini :

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots = \sum_{k=0}^{\infty} \gamma^k r_k \quad (2.2)$$

- Gain en moyenne :

$$R = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=0}^T r_k \quad (2.3)$$

Le paramètre γ joue le rôle d'un taux d'intérêt, on désigne parfois ce paramètre par le terme *coefficient d'amortissement* (discount) qui détermine l'importance des récompenses futures : une récompense reçue k unités de temps plus tard vaut seulement γ^{k-1} de ce qu'elle vaudrait au temps courant. Le gain cumulé est nécessairement fini si jamais $0 < \gamma < 1$ et les r_k sont bornés.

L'importance que l'on accorde aux récompenses futures varie selon la valeur de γ :

- Avec γ proche de zéro ($\gamma \rightarrow 0$), l'agent maximise les récompenses immédiates, et si $\gamma = 0$ l'agent myope apprend à choisir a_t afin de maximiser seulement r_{t+1} .
- Avec γ proche de un ($\gamma \rightarrow 1$), l'agent a un horizon de plus en plus lointain, contrairement à l'agent myope les récompenses futures ont plus d'importance et l'agent devient plus prévoyant.

Il faut noter que le choix de la mesure de gain a une influence déterminante sur la politique optimale (voir la Figure 2.2). Il est donc essentiel de peser soigneusement sa définition avant de lancer le processus d'apprentissage.

Avec $T = 4$ et $\gamma = 0,9$, quelle est la meilleure politique ?	$\sum_{k=0}^T r_k$	$\sum_{k=0}^{\infty} \gamma^k r_k$	$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{k=0}^T r_k$
	6	16	2
	0	59	10
	0	58,4	11

Figure 2.2 – Influence de la mesure de gain sur le choix de la meilleure politique, exemple dû à (Kaelbling *et al.*, 1996)

Dans l'exemple de la Figure 2.2, le choix de la meilleure stratégie, parmi les trois possibles, dépend du critère de gain adopté. Avec $T = 4$ et $\gamma = 0,9$, quelle est la meilleure politique dans l'état initial ? Si l'on considère la première formule de gain, il faut choisir la première politique puisqu'elle conduit au meilleur gain. Pour la deuxième formule de gain, il faut choisir la seconde politique. Et pour la troisième formule, il faut choisir la troisième politique.

Les algorithmes d'apprentissage par renforcement présents dans reste de cette thèse utilisent les fonctions de valeur avec le coefficient d'amortissement γ . Cette formulation est la plus commune dans l'apprentissage par renforcement.

2.2.3 La Fonction d'utilité

La plupart des algorithmes d'apprentissage par renforcement utilisent une fonction d'utilité qui permet, pour une politique π donnée, d'estimer le gain à long terme (les récompenses futures) en partant de l'état s si l'on suit la politique considérée. Les fonctions d'utilité sont définies par l'équation (2.4), où E désigne l'espérance mathématique calculée sur les probabilités de transition.

$$V^\pi(s) = E_\pi \{R_t | s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \quad (2.4)$$

Une propriété fondamentale des fonctions d'utilité est largement utilisée par les algorithmes de programmation dynamique et par l'apprentissage par renforcement. Il s'agit d'une relation récursive connue sous le nom d'équation de Bellman (Bellman, 1957) et décrite par l'équation (2.5) :

$$\begin{aligned} V^\pi(s) &= E_\pi \{R_t | s_t = s\} \\ &= E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\} \\ &= E_\pi \left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right\} \\ &= \sum_{a \in A} \pi(s, a) \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right\} \right] \\ V^\pi(s) &= \sum_{a \in A} \pi(s, a) \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')] \end{aligned} \quad (2.5)$$

De même, nous pouvons définir une fonction d'utilité $Q^\pi(s, a)$ qui permet d'estimer le gain à long terme en partant de l'état s avec l'action a et en suivant la politique π . D'une façon similaire, nous arrivons à :

$$\begin{aligned} Q^\pi(s, a) &= E_\pi \{R_t | s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\} \\ Q^\pi(s, a) &= \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma Q^\pi(s')] \end{aligned} \quad (2.6)$$

2.2.4 Les Politiques optimales

Résoudre un problème d'apprentissage par renforcement signifie trouver une politique qui apporte le gain le plus important possible.

Nous pouvons comparer les politiques en utilisant un ordre partiel sur les fonctions d'utilité :

$$\pi \geq \pi' \Leftrightarrow \forall s, V^\pi(s) \geq V^{\pi'}(s) \quad (2.7)$$

Cet ordre partiel permet de définir la fonction d'utilité de la politique optimale que l'apprentissage par renforcement va chercher à estimer :

$$V^*(s) = \max_{\pi} V^\pi(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \quad (2.8)$$

Soit, pour la fonction Q :

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a')] \quad (2.9)$$

À partir de la connaissance de V^* , nous pouvons construire une politique optimale :

$$\pi^*(s) = \arg \max_a \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \quad (2.10)$$

Ou, si l'on utilise la fonction Q :

$$\pi^*(s) = \arg \max_a Q^*(s, a) \quad (2.11)$$

La plupart des méthodes d'apprentissage par renforcement calculent la fonction V^* ou Q^* et déduisent la politique optimale.

2.3 Exploration et exploitation

L'une des grandes différences entre l'apprentissage par renforcement et les autres types d'apprentissage est l'interaction avec l'environnement. L'algorithme d'apprentissage par renforcement utilise cette interaction pour évaluer les actions exécutées.

Il faut donc prévoir un processus d'exploration de l'espace d'états et d'actions afin de trouver la politique optimale.

Cette situation donne lieu au problème du compromis exploration/exploitation, où il faut en même temps chercher la politique optimale et avoir le meilleur gain possible (Sutton et Barto, 1998) (Russell et Norvig, 1995). Ce type de problème a été bien étudié dans le

domaine de la théorie statistique de la décision sous le nom de problème du bandit manchot¹. (Berry et Fristedt, 1985) (Auer *et al.*, 2002)

Le bandit manchot est un problème dont le nom repose sur une analogie avec une machine à sous traditionnelle (bandit manchot). Tirer le levier fournit une récompense suivant la distribution de probabilité associée à la machine concernée. A chaque pas, l'agent a donc le choix entre des actions exploratoires, et des actions dont le gain est déjà connu. On considère que le joueur a N machines à sa disposition et qu'il a un nombre fini M d'essais.

L'objectif du joueur est donc de maximiser la somme des récompenses obtenues par des actions successives. On suppose classiquement que le joueur n'a aucune connaissance initiale à propos du comportement des machines. En conséquence, le joueur doit choisir, à chaque étape, d'exploiter la machine qui a la plus grande espérance de gains ou d'explorer pour avoir plus d'information sur l'espérance des autres machines.

Les algorithmes de sélection d'action développés pour le bandit manchot s'appliquent à l'apprentissage par renforcement dans des PDM à plusieurs états, puisqu'on y retrouve le même dilemme. En effet, il faut trouver le bon compromis entre la maximisation des récompenses basées sur la connaissance déjà acquise, et l'essai d'autres actions pour accroître la connaissance de l'environnement.

Les sous-sections suivantes présentent deux stratégies possibles pour le choix des actions : ϵ -gloutonne et boltzmannienne. Ces deux stratégies sont les plus couramment utilisées par l'apprentissage par renforcement. D'autres solutions sont proposées dans la littérature, mais elles ne seront pas détaillées dans le présent document. Un exemple est la stratégie optimiste qui simplifie la gestion du dilemme entre exploration et exploitation en initialisant toutes les espérances de récompense à une valeur optimiste égale au maximum des récompenses immédiates atteignables dans l'environnement (Brafman et Tennenholtz, 2003).

2.3.1.1 La stratégie ϵ -gloutonne

La stratégie ϵ -gloutonne² est une des premières et des plus simples solutions pour le problème de l'exploration. Elle utilise une action dite gloutonne (qui est l'action la meilleure évaluée par la politique courante), une distribution uniforme de probabilités et un $\epsilon \in [0,1]$.

Cette stratégie choisit l'action gloutonne avec une probabilité $1 - \epsilon$ du temps, et une autre action est aléatoirement choisie (en utilisant une distribution uniforme) avec une probabilité ϵ .

¹ « Bandit problem » ou « n -armed bandit problem » en anglais

² « ϵ -greedy » en anglais

Le choix de l'action prochaine a_t est donc :

$$a_t = \begin{cases} \arg \max_a Q(s, a) & \text{avec probabilité } 1 - \varepsilon \\ \text{action aléatoire} & \text{avec probabilité } \varepsilon \end{cases} \quad (2.12)$$

2.3.1.2 La stratégie boltzmannienne

Pour la stratégie boltzmannienne, la prochaine action a_t est choisie selon la distribution de Boltzmann, décrite par l'équation (2.13):

$$P(a_t) = e^{Q(s, a_t)/T} \cdot \left[\sum_{a \in A} e^{Q(s, a)/T} \right]^{-1} \quad (2.13)$$

où T , par analogie avec la physique, est la température associée à la distribution. Quand T est élevée, la distribution est presque uniforme ; quand $T \rightarrow 0$, on se rapproche de la stratégie 0-gloutonne.

En pratique, on fait décroître la température, ce qui permet de moduler exploration et exploitation sans distinguer explicitement ces deux phases.

2.4 Les principales méthodes

Cette section décrit trois catégories de méthodes fondamentales pour résoudre le problème de l'apprentissage par renforcement: la programmation dynamique, les méthodes Monte Carlo et les méthodes de différences temporelles.

Chaque classe de méthode a ses avantages et ses inconvénients. La programmation dynamique est mathématiquement bien développée, mais elle a besoin d'un modèle complet et assez précis de l'environnement. La méthode Monte Carlo n'a pas besoin de la définition complète d'un modèle et est conceptuellement simple, mais elle n'est pas adaptée au calcul incrémental pas par pas sans avoir à attendre la fin de l'épisode pour apprendre. Enfin, les méthodes de différences temporelles ne nécessitent pas un modèle de l'environnement et sont entièrement incrémentales, mais elles sont plus complexes à analyser. Ces méthodes sont également différentes en ce qui concerne leur efficacité et leur temps de convergence. Dans la section 2.5 nous allons montrer comment ces méthodes peuvent être combinées de façon à obtenir les meilleures caractéristiques de chacune.

2.4.1 Programmation dynamique

Le terme Programmation Dynamique fait référence à une collection d'algorithmes qui sont employés pour calculer des politiques optimales étant donné un modèle complet de l'environnement tel que décrit par le processus de décision Markovien (PDM). La

programmation dynamique cherche à estimer la fonction d'utilité optimale V^* afin d'en déduire une politique optimale (Bellman, 1957) (Bertsekas et Tsitsiklis, 1996).

Les algorithmes classiques de programmation dynamique ont une utilité limitée en apprentissage par renforcement à cause de leur hypothèse de connaissance parfaite du modèle. La plupart de ces méthodes sont incrémentales : elles s'approchent petit à petit de l'utilité optimale par itérations successives. Parmi ces méthodes l'algorithme d'itération de la valeur (Sutton et Barto, 1998) est l'une des plus simples et des plus répondues. Les méthodes de programmation dynamique ne sont pas adaptées à des problèmes de très grande taille, dans la pratique on peut appliquer ces algorithmes à des PDM ayant jusqu'à l'ordre de 10^6 états, en utilisant les ordinateurs actuels. Pour traiter les PDM de plus grande taille, on peut par exemple se tourner vers des méthodes de programmation dynamique approchée (Bertsekas et Tsitsiklis, 1996).

La programmation dynamique est ainsi plus adaptée à des problèmes de planification (ou problèmes d'optimisation), vu l'exigence d'une connaissance parfaite du modèle de l'environnement, des fonctions de transition et de renforcement, et du déroulement de l'*apprentissage « hors-ligne »* (sans interagir avec leur environnement).

2.4.2 Les méthodes de Monte Carlo

Les méthodes Monte Carlo sont basées sur l'idée similaire à la programmation dynamique : l'estimation de la fonction d'utilité afin d'améliorer la politique. Toutefois, contrairement à la programmation dynamique qui utilise un modèle complet de l'environnement, les méthodes Monte Carlo ont recourt à des expériences réalisées dans l'environnement (*apprentissage « en ligne »*). Les méthodes Monte Carlo reposent donc sur l'expérience plutôt que sur une connaissance parfaite de l'environnement, cela élargit le champ d'application de cette méthode (Michie et Chambers, 1968) (Rubinstein, 1981) (Robert et Casella, 2004).

(Filliat, 2004) présente une bonne synthèse de l'évaluation de la politique pour les méthodes Monte-Carlo. L'estimation de la fonction d'utilité est faite à partir d'un ensemble de séquences “état-action-récompenses-état-action ...” réalisées par l'agent. Pour les états de ces séquences, il est alors possible d'estimer V simplement par la moyenne des revenus :

$$V(s) = \frac{1}{N(s)} \sum Revenus(s) \quad (2.14)$$

où $N(s)$ est le nombre de séquences où apparaît s et $Revenus(s)$ est le revenu après la première visite de l'état s , c'est-à-dire la somme des récompenses avec intérêt après cette visite.

L'algorithme de la Figure 2.3 montre comment évaluer la fonction V^π avec la méthode Monte Carlo. Elle génère des épisodes avec la politique π , garde les résultats obtenus en chaque épisode et utilise la moyenne de ces résultats pour déterminer $V(s)$.

```

1 :   Initialiser :
2 :    $\pi \leftarrow$  une politique pour être évaluée
3 :    $V \leftarrow$  une fonction état-valeur arbitraire
4 :    $Revenus(s) \leftarrow$  une liste vide, pour tout  $s \in S$ 
5 :
6 :   Repeat
7 :       Générer un épisode en utilisant  $\pi$ 
8 :       Pour chaque état  $s$  de l'épisode :
9 :            $R \leftarrow$  return en suivant la première occurrence de  $s$ 
10 :          Ajouter  $R$  à  $Revenus(s)$ 
11 :           $V(s) \leftarrow$  moyenne( $Revenus(s)$ )
12 :   until toujours

```

Figure 2.3 – Méthode Monte Carlo pour évaluer V^π .

Avec la possibilité d'évaluer de la politique π (algorithme de la Figure 2.3) et la possibilité d'améliorer la politique en utilisant l'équation :

$$\pi'(s) = \arg \max_a Q^\pi(s, a)$$

$$\pi'(s) = \arg \max_a E\{r_{t+1} + \mathcal{W}^\pi(s_{t+1}) | s_t = s, a_t = a\}$$

$$\pi'(s) = \arg \max_a \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \mathcal{W}^\pi(s')], \text{ il est facile de voir un processus}$$

itératif d'amélioration de politique donné par la séquence :

$$\pi_0 \xrightarrow{E} Q^{\pi_0} \xrightarrow{A} \pi_1 \xrightarrow{E} Q^{\pi_1} \xrightarrow{A} \pi_2 \xrightarrow{E} \dots \xrightarrow{A} \pi^* \xrightarrow{E} Q^{\pi^*}, \quad \text{où } \xrightarrow{E} \text{ et } \xrightarrow{A}$$

dénote une phase d'évaluation et \xrightarrow{A} une phase d'amélioration. Cette procédure itérative converge en un nombre fini d'itérations vers la politique optimale si la politique est représentée par un PDM à nombre d'états fini (Cornuéjols et Miclet, 2002).

Un algorithme simple qui réalise ce processus est illustré par la Figure 2.4.

```

1 : Initialiser :
2 :  $\pi(s) \leftarrow$  une politique pour être évaluée
3 :  $V(s) \leftarrow$  une fonction état-valeur arbitraire
4 :  $Revenus(s,a) \leftarrow$  une liste vide
5 :
6 : Repeat
7 :   Générer un épisode en utilisant une stratégie d'exploration et  $\pi$ 
8 :   Pour chaque paire  $s,a$  de l'épisode :
9 :      $R \leftarrow$  return en suivant la première occurrence de  $s,a$ 
10:    Ajouter  $R$  à  $Revenus(s)$ 
11:     $Q(s,a) \leftarrow$  moyenne( $Revenus(s,a)$ )
12:    Pour chaque état  $s$  de l'épisode :
13:       $\pi(s) \leftarrow \arg \max_a Q(s,a)$ 
14: until toujours

```

Figure 2.4 – Méthode Monte Carlo simple pour estimer $Q^*(s,a)$.

Pour que cet algorithme marche il faut considérer un mécanisme d'exploration qui va permettre qu'un état (ou état-action) soit choisi un nombre infini de fois. Cela veut dire que si l'environnement et la politique sont déterministes, il faut considérer l'hypothèse des « départs exploratoires¹ » : où chaque état (ou état-action) a une probabilité non nulle d'être le point de départ d'un épisode. Cela assure que tous les états de l'environnement vont être explorés dans un certain temps.

La programmation dynamique permet d'utiliser les estimations du gain des états ultérieurs pour évaluer et mettre à jour l'utilité d'un état actuel (cette procédure est appelée « *bootstrap* ») mais nécessite un modèle complet et précis de l'environnement. Les méthodes de Monte-Carlo permettent d'apprendre directement à partir des expériences, sans aucun modèle de l'environnement, mais font pas le *bootstrap*.

Dans les sections suivantes sera présentée la méthode d'apprentissage par *différences temporelles* (TD²) qui réunit les deux avantages cités ci-dessus. Nous présentons en particulier deux variantes de cette méthode : *Sarsa* et *Q-learning*.

2.4.3 Les méthodes de différences temporelles

Les méthodes d'apprentissage par différences temporelles réunissent le *bootstrap* de la programmation dynamique avec l'avantage du calcul incrémental pas à pas des méthodes de Monte-Carlo, ainsi les méthodes TD permettent l'apprentissage sans modèle de l'environnement et, évaluent et mettent à jour l'utilité d'un état en fonction des estimations des états successeurs sans avoir à attendre la fin de la simulation (Sutton et Barto, 1998).

¹ De l'anglais « Exploring Starts »

² TD de l'anglais « Temporal-Difference »

L'évaluation de la politique va donc se faire à partir d'expériences comme pour la méthode de Monte-Carlo qui utilise une moyenne pour estimer l'utilité d'un état. Cette estimation mise sous forme itérative conduit à l'équation suivante :

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \quad (2.15)$$

avec :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^p r_{t+p+1} \quad (2.16)$$

On sait d'après les équations de la section précédente que $E[R_t | s_t] = E[r_t + \gamma V(s_{t+1}) | s_t]$, donc en ajoutant l'idée d'utiliser l'estimation du gain à partir de la récompense suivante et de l'utilité de l'état suivant, au lieu d'utiliser la somme des récompenses, on obtient :

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2.17)$$

Cette mise à jour se fait naturellement de manière incrémentale, au fur et à mesure des expériences de l'agent. Les TD sont une famille d'algorithmes d'apprentissage par renforcement, la Figure 2.5 montre l'algorithme TD le plus simple (TD(0)). D'autres algorithmes comme le *Sarsa* et le *Q-learning* seront présentés dans les sections subséquentes.

1 :	Initialiser $V(s)$ arbitrairement, π la politique à évaluer
2 :	Repeat (pour chaque épisode) :
3 :	Initialise s
4 :	Repeat (pour chaque pas de l'épisode) :
5 :	$a \leftarrow$ action donné par π pour l'état s
6 :	Exécuter a , recueillir r et l'état suivant s'
7 :	$V(S) \leftarrow V(s) + \alpha [r + \gamma V(s') - V(s)]$
8 :	$s \leftarrow s'$
9 :	until s soit terminal

Figure 2.5 - TD(0) tabulaire pour estimer $V^\pi(s)$

Apprendre la fonction V ne suffit pas à en déduire la politique optimale lorsque l'on n'a pas de modèle. A la place, il faut apprendre la fonction Q .

2.4.3.1 Sarsa

Si l'on veut considérer les transitions entre des paires état/action et apprendre l'utilité de ces paires à la place de l'utilité des états, nous sommes devant l'apprentissage de la fonction Q qui peut avoir une mise à jour équivalente à l'équation (2.17) donné par :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2.18)$$

Cette méthode d'apprentissage par renforcement est connue sous le nom de Sarsa¹. On dit qu'elle est « *on-policy* » ou qu'elle suit la politique, car elle utilise l'action a_{t+1} qui dépend de la politique pour mettre à jour les valeurs de la fonction Q . C'est-à-dire que l'agent apprend en suivant la politique qui lui est attribuée à l'instant t .

```

1 : Initialiser  $Q(s,a)$  arbitrairement
2 : Repeat (pour chaque épisode) :
3 :   Initialise  $s$ 
4 :   Choisir  $a$  à partir de  $s$  en utilisant une politique dérivé de  $Q$  (p.ex.  $\epsilon$ -gloutonne)
5 :   Repeat (pour chaque pas de l'épisode) :
6 :     Exécuter  $a$ , recueillir  $r$  et l'état suivant  $s'$ 
7 :     Choisir  $a'$  à partir de  $s'$  en utilisant une politique dérivé de  $Q$  (p.ex.  $\epsilon$ -gloutonne)
8 :      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$ 
9 :      $s \leftarrow s'; a \leftarrow a'$ ;
10 :  until  $s$  soit terminal

```

Figure 2.6 – SARSA : un algorithme TD « *on-policy* »

2.4.3.2 *Q-learning*

Une des plus importantes réussites dans l'apprentissage par renforcement a été le développement de l'algorithme *Q-learning* (Watkins, 1989). Cet algorithme est la variante « *off-policy* » de *Sarsa* qui utilise le maximum de la fonction sur les actions suivantes au lieu de l'action qui sera effectivement exécutée :

$$Q(S_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2.19)$$

Cet algorithme fait Q converger vers Q^* d'une façon indépendante de la politique suivie, cela simplifie nettement l'analyse de l'algorithme et l'obtention de preuves formelles de convergence (Watkins et Dayan, 1992).

```

1 : Initialiser  $Q(s,a)$  arbitrairement
2 : Repeat (pour chaque épisode) :
3 :   Initialise  $s$ 
4 :   Repeat (pour chaque pas de l'épisode) :
5 :     Choisir  $a$  à partir de  $s$  en utilisant une politique dérivé de  $Q$  (p.ex.  $\epsilon$ -gloutonne)
6 :     Exécuter  $a$ , recueillir  $r$  et l'état suivant  $s'$ 
7 :      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a') - Q(s,a)]$ 
8 :      $s \leftarrow s'$ 
9 :   until  $s$  soit terminal

```

Figure 2.7 – *Q-learning* : un algorithme TD « *off-policy* »

¹ Dont le nom vient de la tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$

2.5 L'unification de Monte-Carlo et des méthodes de différences temporelles

Les algorithmes de différences temporelles vues dans la section 2.4.3 ne font que la mise à jour, à chaque instant t , de l'utilité estimée de l'état (ou état-action) à instant $t-1$. Cela signifie une mise à jour à la fois, cependant il y a d'autres méthodes avec un mécanisme de mise à jour plus efficace.

Le premier exemple est l'utilisation de n pas, qui conduit à la méthode des différences temporelles à n pas. Dans cette méthode, le gain est estimé par une équation de la forme :

$$R_t = R_t^n = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^n r_{t+n+1} \mathcal{N}(s_{t+n+1}) \quad (2.20)$$

La mise à jour est donc définie par l'équation :

$$V(S_t) \leftarrow V(s_t) + \alpha [R_t^n - V(s_t)] \quad (2.21)$$

Il est également possible d'estimer R_t en utilisant des moyennes pondérées de R_t^n et un cas particulier intéressant de cette dernière méthode est l'utilisation d'une pondération exponentielle qui fait décroître l'importance des expériences au fur et à mesure de leur éloignement dans le temps :

$$R_t = (1-\lambda) \sum_{i=1}^{\infty} \lambda^i R_t^i \quad (2.22)$$

Ce cas particulier est intéressant, car il peut s'appliquer simplement en utilisant les valeurs passées des récompenses, au lieu des valeurs futures (il a été démontré que les mises à jour sont les mêmes (Sutton et Barto, 1998)). On utilise pour cela une valeur auxiliaire appelée trace d'éligibilité que l'on définit de la manière récursive suivante :

$$e_t(s) = \begin{cases} \gamma e_{t-1}(s) & \text{si } s \neq s_t \\ \gamma e_{t-1}(s) + 1 & \text{si } s = s_t \end{cases} \quad (2.23)$$

Néanmoins, l'agent peut passer plusieurs fois par le même état avant d'arriver à la fin, et cela peut produire des problèmes quand des longues traces cumulatives (2.23) sont employées car ils vont produire des valeurs assez grandes pour l'algorithme d'apprentissage par renforcement.

Sutton et Barto (Sutton et Barto, 1998) ont donc proposé une légère modification de la mise à jour des traces qui consiste à remplacer l'accumulation par un remplacement, conformément à l'équation suivante :

$$e_t(s) = \begin{cases} \gamma e_{t-1}(s) & \text{si } s \neq s_t \\ 1 & \text{si } s = s_t \end{cases} \quad (2.24)$$

Il a été également démontré que l'utilisation des traces d'éligibilité conduit à une amélioration significative des performances de l'algorithme d'apprentissage par renforcement.

En tenant en compte les traces d'éligibilité, la mise à jour est donc faite d'une manière proportionnelle à l'éligibilité pour chaque état, selon l'équation :

$$V(s_t) \leftarrow V(s_t) + \alpha e(s_t)[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2.25)$$

L'équation (2.25) remplace la mise à jour d'un état en utilisant des récompenses futures par la mise à jour des états passés en utilisant la récompense courante. Cela donne au final l'algorithme $\text{TD}(\lambda)$ décrit par la Figure 2.8.

```

1 :   Initialiser  $V(s)$  aléatoirement et  $e(s) = 0$  pour tout  $s \in S$ 
2 :   Repeat
3 :       Initialise  $s$ 
4 :       for all pas de l'épisode do
5 :            $a \leftarrow \pi(s)$ 
6 :           Exécuter  $a$ , recueillir  $r$  et l'état suivant  $s'$ 
7 :            $\delta \leftarrow r + \gamma V(s') - V(s)$ 
8 :            $e(s) \leftarrow e(s) + 1$  (ou  $e(s) \leftarrow 1$ )
9 :           for all  $s$  do
10:               $V(s) \leftarrow V(s) + \alpha \delta e(s)$ 
11:               $e(s) \leftarrow \gamma \lambda e(s)$ 
12:           end for
13:            $s \leftarrow s'$ 
14:       end for
15:   until épisode ne pas fini (ou  $s$  soit un état final)

```

Figure 2.8 - Algorithme $\text{TD}(\lambda)$

Il suffit de changer l'algorithme $\text{TD}(\lambda)$ (Figure 2.8) d'une manière élémentaire pour trouver les extensions $\text{Sarsa}(\lambda)$ et $Q(\lambda)$.

2.6 Apprentissage dans des espaces de grande taille

Jusqu'à présent, nous n'avons pas faire attention à la taille de l'espace d'états et de l'espace d'actions. Nous avons implicitement supposé qu'il était possible d'indexer les états et les actions un utilisant l'ordinateur, mais cela n'ai pas toujours possible. Pour les problèmes de grande taille il est impossible de représenter les différentes fonctions (par exemple la fonction V) par des tables de valeurs. L'utilisation des tables de valeurs est simple et intuitive cependant elle est limitée aux tâches qui ont un petit nombre d'états et d'actions. De plus, dans de nombreuses applications, l'espace des états, et parfois aussi celui des actions, est continu, rendant ainsi impossible l'usage direct de tables.

Le problème n'est pas seulement lié à la mémoire nécessaire pour les grandes tables, mais également au temps et la quantité d'information (expérience acquise grâce aux itérations avec l'environnement) nécessaire pour les remplir. Les algorithmes de mise à jour de tables font une utilisation peu efficace de l'information glanée en cours d'expérience. En effet, alors qu'il est fréquent que des états similaires aient des valeurs d'utilité et des actions optimales attachées similaires, la modification d'une valeur dans la table n'entraîne pas celle des autres. Dans ces conditions, l'utilisation des tables semble peu efficace.

Le besoin d'une représentation à la fois plus compacte et permettant une utilisation plus efficace de l'information nous amène à des techniques de généralisation de l'espace des états, et éventuellement de l'espace des actions. L'utilisation de ces techniques rend possible la généralisation de l'information obtenue par un état et une action observée.

Le type de généralisation dont nous avons besoin est appelé approximation de fonction parce qu'il prend des valeurs de la fonction souhaitée (par ex., une fonction d'utilité) afin d'essayer de généraliser à partir de ces exemples et de construire une approximation de la fonction entière. L'approximation de fonction est en apprentissage supervisé appelé « *régression* » et est un point très étudié.

2.6.1 Généralisation par approximation de fonctions

Au lieu de stocker toutes les valeurs de la fonction V_t ou Q_t dans une table, nous pouvons la représenter de façon approchée sous la forme d'une fonction paramétrée par un vecteur $\vec{\theta}_t$. Cela veut dire que la fonction estimée V_t dépend totalement du vecteur $\vec{\theta}_t$ qui varie de t à $t+1$.

Par exemple, V_t pourrait être réalisée par un réseau de neurones, où le vecteur $\vec{\theta}_t$ représente les poids de connections synaptiques (Sutton et Barto, 1998). Un autre exemple serait l'utilisation d'un arbre de décision à profondeur bornée ; dans ce cas-là le vecteur $\vec{\theta}_t$ décrit les branchements de l'arbre appris (Degris, 2007). Typiquement, le nombre de paramètres (le nombre de éléments de $\vec{\theta}_t$) est largement inférieur au nombre d'états, et l'altération d'un paramètre change la valeur estimée de plusieurs états. Par conséquent, quand l'estimation d'un seul état est actualisée, cette mise à jour est généralisée et peut affecter la valeur des états voisins.

Des nombreux travaux concernant des expériences d'apprentissage par renforcement avec généralisation ont été publiés : (Boyan et Moore, 1995) ont utilisé des méthodes des plus proches voisins pour une approche d'itération de valeur ; (Lin, 1991; Lin, 1992) a mis en œuvre un réseau connexionniste avec apprentissage par rétropropagation de gradient pour

apprendre la fonction d'utilité $Q(s, a)$ dans le Q-learning ; (Watkins, 1989), toujours dans le cadre du Q-learning, a utilisé la technique CMAC¹ due à Albus (Albus, 1975; Albus, 1981) et a été suivi par d'autres chercheurs ; (Tesauro, 1990; Tesauro, 1992; Tesauro, 1995; Tesauro, 2002) a utilisé un réseau connexionniste pour apprendre la fonction $V(s)$ pour le jeu de Backgammon ; (Zhang et Dietterich, 1995) ont utilisé un réseau connexionniste dans une technique de différence temporelle TD(λ) pour apprendre des stratégies d'ordonnancement de tâches pour des ateliers.

Néanmoins, bien que les méthodes de généralisation soient largement utilisées par la communauté, elles présentent quelques inconvénients. Dans les environnements discrets avec maintien de tables de valeur, il existe des garanties que toute opération qui met à jour la fonction d'utilité (selon les équations de Bellman) ne peut que réduire l'erreur entre l'utilité courante estimée et l'utilité optimale. Or de telles garanties n'existent plus avec des techniques de généralisation (Boyan et Moore, 1995).

Une autre cause potentielle de problème vient du fait que les méthodes d'exploration classiques, comme la méthode ϵ -gloutonne, ne soient pas adaptées à cette situation. Cela est encore une question ouverte et certains chercheurs étudient ce sujet pour mieux comprendre les interactions entre échantillonnage, apprentissage et amélioration de politique (Cornuéjols et Miclet, 2002).

Malgré ces difficultés, l'approximation de fonction basée sur la descente de gradient reste très utilisée. Cette descente de gradient tend à réduire l'écart quadratique entre les valeurs estimées à l'instant t et la cible courante, par exemple les valeurs rapportées par la méthode de différences temporelles. Nous détaillons maintenant cette approche.

2.6.1.1 Cas général

Pour toute fonction V_t dérivable par rapport à $\vec{\theta}_t$, la mise à jour de ce paramètre peut se faire avec la formule suivante :

$$\theta_{t+1} = \theta_t + \alpha [v_t - V_t(s_t)] \nabla_{\theta_t} V_t(s_t) \quad (2.26)$$

où, v_t représente la cible, c'est-à-dire l'estimation courante de $V^\pi(s_t)$, et α est un pas d'apprentissage décroissant. Cette méthode peut être par exemple être utilisée avec les réseaux de neurones.

¹ CMAC de l'anglais « Cerebellar Model Articulatory Controller »

2.6.1.2 Cas linéaire

Il est très fréquent de représenter la fonction de valeur sous la forme d'une combinaison linéaire de n fonctions de base pour approximer la fonction cible. L'ensemble des fonctions de base $\Phi(s) = \{\phi_1(s), \dots, \phi_n(s)\}$ est défini sur l'espace S des états. L'approximation de la fonction d'utilité correspond alors à la formule :

$$V_t(s) = \vec{\theta}_t^T \Phi(s) = \sum_{i=1}^n \theta_t(i) \phi_i(s) \quad (2.27)$$

Dans ce cas, le gradient de la fonction V_t par rapport à $\vec{\theta}_t$ devient : $\nabla_{\theta_t} V_t(s) = \Phi(s)$.

Cette règle a l'avantage d'être simple et de conduire à un seul optimum (Cornuéjols et Miclet, 2002). Comme de plus elle est assez efficace en terme de données à stocker et de calculs, elle a la faveur de nombreux chercheurs et praticiens.

Une approche duale consiste à apprendre non pas la combinaison de fonctions de base, mais les fonctions de base elles-mêmes. Cela consiste à apprendre des voisinages dans l'espace (Sutton et Barto, 1998).

Dans ce cadre des méthodes d'approximation linéaire il existe les *techniques par couverture de voisinages uniformes*. L'idée est d'associer un voisinage à chaque état dont la valeur est connue, par exemple une boule (voir Figure 2.9). La valeur d'un état inconnu est décidée en fonction des voisinages, les boules, dont il fait partie. On utilise une fonction linéaire définissant les voisinages (les boules) dont le point fait partie. On peut modifier la densité des boules, ou bien leur forme, ce qui revient alors à rendre non linéaire l'approximation.

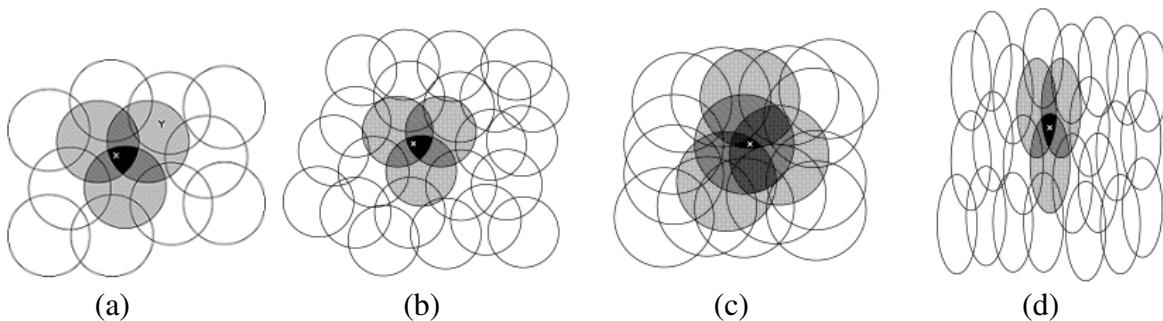


Figure 2.9 - La valeur cherchée au point X (par exemple $V(X)$) dépend des valeurs des points Y dont les voisinages incluent X (Sutton et Barto, 1998).

Codage en carreaux (*Tile Coding*)

Une méthode d'approximation linéaire par couverture de voisinages uniformes largement utilisée en apprentissage par renforcement est la méthode CMAC (Albus, 1975; Albus, 1981). Cette méthode représente et partitionne les points de l'espace sous forme de grille. On utilise

donc des carreaux à la place des boules de la Figure 2.9, cela est particulièrement bien adapté au calcul discret réalisé par les ordinateurs et efficace pour l'apprentissage en ligne.

La méthode des CMAC consiste à définir C grilles d'un espace mémoire de K régions. Pour un état $s \in S$, la valeur recherchée $F(s)$ (F est la fonction complexe que l'on veut représenter, par exemple V) est la somme des valeurs des C régions contenant s . Pour chaque état, chaque grille contient en effet une région qui le couvre. Ces régions sont définies de telle manière que des états qui sont proches aient des régions en commun. Ainsi, la fonction résultante F fait apparaître une généralisation dans les voisinages de l'espace d'états S . La Figure 2.10 montre le recouvrement de l'espace d'états par C grilles de K régions. Nous pouvons remarquer la façon dont les grilles sont décalées pour créer des combinaisons distinctes des paramètres C pour les différents points de l'espace.

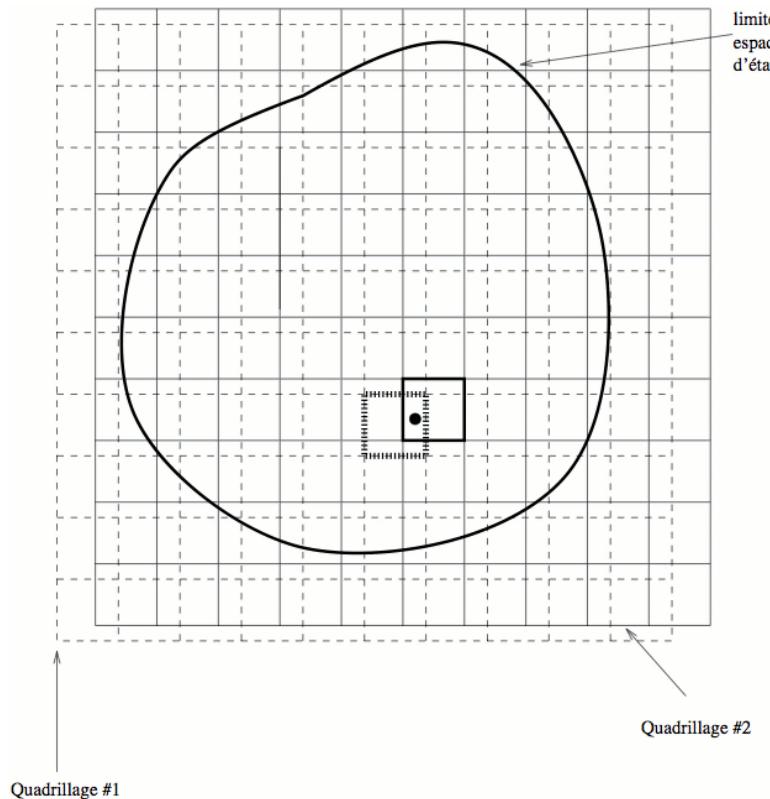


Figure 2.10 - Un CMAC avec $C = 2$ et $K = 100$.

Un autre point important à considérer au moment de l'utilisation du CMAC est l'emploi des techniques de hachage pour réduire l'espace de mémoire nécessaire qui dépend du nombre de partitions utilisées et de la manière dont elles sont fractionnées.

2.7 Conclusion

Ce chapitre n'a traité qu'une toute petite partie de l'apprentissage par renforcement. Nous avons choisi ce qu'il est plus pertinent pour le reste du manuscrit. Il existe, par exemple, un certain nombre de méthodes d'apprentissage basés sur de modèles. Mais ces méthodes ne sont pas extensibles pour les problèmes grande taille avec l'espace continu. Ces derniers type problèmes sont le centre d'intérêt des agents qui réalisent des tâches du monde réel (Taylor *et al.*, 2008).

Chapitre 3

Etat de l'art du transfert de connaissance

Après avoir vu les principes de l'apprentissage par renforcement, nous présentons les bases et la problématique du transfert de connaissance en apprentissage par renforcement. Nous aborderons dans ce chapitre les travaux liés aux techniques de transfert que nous avons créé et expérimenté qui, elles-mêmes, seront présentées dans le Chapitre 4.

3.1 Bases du transfert de connaissance en apprentissage par renforcement

L'apprentissage par renforcement est progressivement devenu un sujet très actif et prometteur de l'apprentissage automatique et de l'intelligence artificielle. Mais, lorsque l'espace d'état/actions est grand, les méthodes traditionnelles d'apprentissage par renforcement n'ont pas d'autre possibilité que l'explorer de façon exhaustive. Comme le temps d'exploration

croît avec la taille de l'espace d'état ou d'action cela devient un problème car l'apprentissage va tarder à converger.

Afin de palier ce problème les concepteurs d'algorithme d'apprentissage par renforcement passent beaucoup de temps à raffiner la représentation de l'espace d'état (en cherchant à réduire sa taille en ne représentant que les caractéristiques pertinentes du problème) et les paramètres (compromis exploration/exploitation, vitesse d'apprentissage et influence des actions passées) afin d'optimiser et accélérer le processus d'apprentissage.

Néanmoins, ce travail de conception ne suffit souvent pas à résoudre des problèmes d'apprentissage de grande taille. En effet, les algorithmes d'apprentissage standards comme le *Q-learning* ou *R-max* exploreront l'espace d'état de façon systématique, et le temps de calcul sera d'autant plus grand que cet espace est important. Par exemple, il est connu que le temps de convergence de *R-max* vers une politique presque optimale est quadratique en fonction de la taille de l'espace d'états.

Or, pour de nombreux problèmes d'apprentissage, on dispose déjà de connaissances pouvant aider à les résoudre. L'utilisation de cette connaissance pour guider l'exploration est nommée « *transfert de connaissance* ». Si la connaissance est suffisamment précise, on peut espérer guider l'exploration de telle sorte que seule une infime partie de l'espace d'état soit effectivement exploré.

Nous illustrons maintenant cette notion de transfert de connaissance sur quelques exemples.

Le jeu de go. Résoudre le problème du jeu de go en apprentissage par renforcement fait appel à un espace d'état énorme. Néanmoins, de nombreuses heuristiques sont connues. Par exemple, contrôler les coins du goban peut donner un avantage stratégique important. Cette heuristique peut être utilisée pour guider l'exploration que l'on utilise un algorithme de type Monte Carlo ou autres.

Sortir d'une zone encombrée. Supposons qu'un agent veuille sortir d'une zone pleine d'obstacles (Figure 3.1(a)).

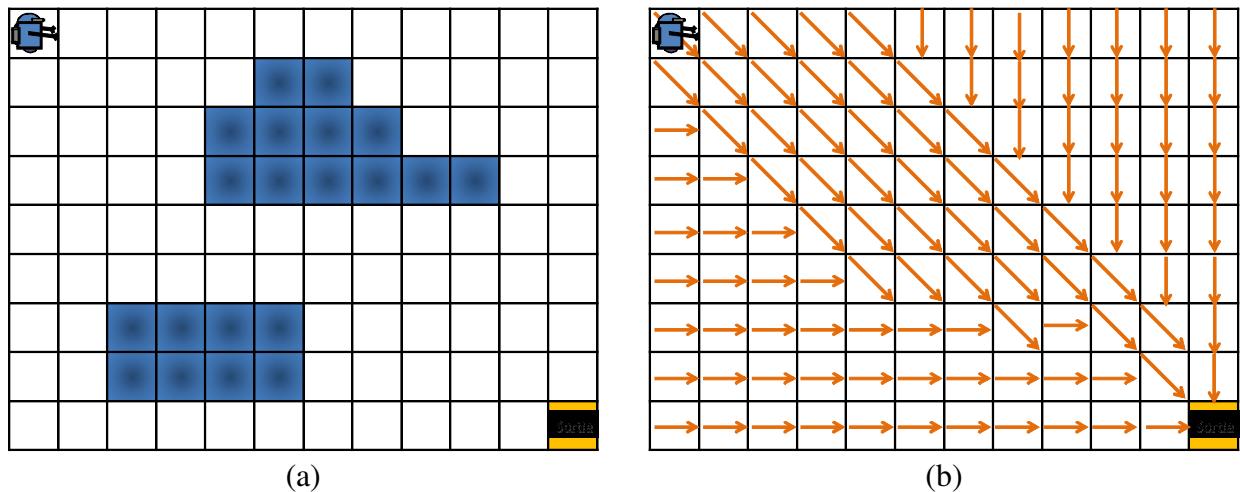


Figure 3.1 – (a) Sortir d'une zone encombrée ; (b) Politique apprise sans les obstacles.

Supposons qu'il ait déjà appris à sortir de cette zone avant que les obstacles y soient placés (Figure 3.1(b)). L'agent pourrait alors utiliser la politique apprise sur cette tâche plus simple pour résoudre la tâche avec obstacles, sans explorer la totalité de l'espace.

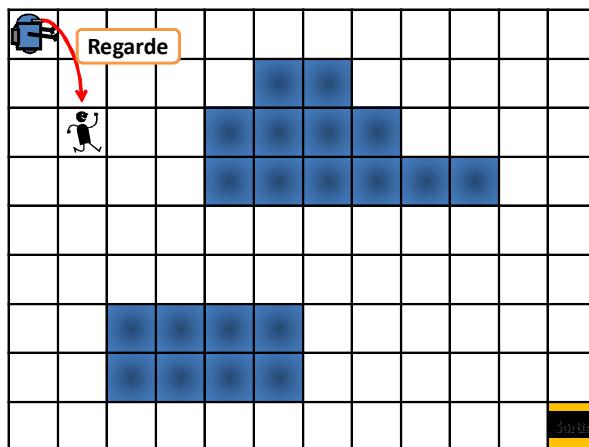


Figure 3.2 – Observation d'un humain.

Dans la même situation, l'agent pourrait observer un autre agent, ou un humain agissant dans le même environnement, et utiliser ces observations pour guider l'exploration (Figure 3.2).

Dans les sections suivantes, nous détaillerons les problématiques et techniques du transfert de connaissance.

3.2 Problématiques du transfert en apprentissage par renforcement

L'objectif principal du transfert de connaissance en apprentissage par renforcement est d'utiliser une connaissance disponible pour accélérer le processus d'apprentissage. Et le point principal de la problématique du transfert de connaissance en apprentissage par renforcement est comment la réutiliser/transférer vers l'agent apprenant.

Comment peut-on réutiliser une connaissance disponible dans un processus d'apprentissage automatique ? Cela est sans doute la question la plus générale du domaine du transfert de connaissance. Mais elle s'accompagne d'autres questions que nous allons développer pour mieux comprendre les problématiques du domaine.

Il y a différentes situations avec différents types de problèmes à affronter au moment du transfert de connaissance. Mais partons de la connaissance que l'on a à disposition. Comme nous l'avons vu sur des exemples, elle peut être de différents ordres : elle peut être un ensemble de trajectoires d'un agent « mentor », sous forme d'une suite d'états explorés, ou une suite d'états actions ; elle peut être une heuristique, ou politique fournie par un mentor qui résoudrait une tâche similaire ou corrélée ; voir même une fonction Q ou V . Cette politique (ou fonction Q/V) peut être représentée de diverses façons : un tableau, une base de règles, un arbre de décision, un réseau de neurones, etc.

Etant donné une connaissance à transférer, est-ce qu'elle peut être utilisée dans son état actuel ou doit-elle être changée vers une autre représentation plus adapté au processus d'apprentissage ? Si oui comment peut-on faire ce changement ? (Nehaniv et Dautenhahn, 1998)

Une fois que la connaissance est disponible d'une manière qui peut être exploitée par les agents apprenants, cette source est-elle entièrement fiable/adaptée au problème que nous voulons résoudre ? Ou y a-t-il des parties qui vont nuire à l'apprentissage automatique ? Il faut savoir donc sélectionner la connaissance à transférer.

Dans certains cas, l'utilisation du transfert de connaissance obtient des résultats inférieurs à ceux de l'apprentissage sans le transfert de connaissance. Cet effet est ce que l'on appelle « transfert négatif » et il est par exemple trouvé dans les expérimentations de (Mehta *et al.*, 2008) avec son exemple de jeu de stratégie en temps réel simplifié.

Une autre situation possible est d'avoir plusieurs sources de connaissance disponibles pour le transfert. Dans ce cas, il y a d'autres points à considérer : quelle source doit-on choisir ? Faut-il en choisir une seule ou en combiner plusieurs ?

Il y a encore des cas où nous n'avons pas de connaissance à transférer, mais il se trouve qu'il est plus avantageux de créer cette connaissance à partir de l'apprentissage de tâches plus simples. Autrement dit, il est plus rapide de simplifier la tâche originale, apprendre à résoudre cette tâche plus simple, pour ensuite utiliser cette connaissance dans le processus d'apprentissage de la tâche originale, que faire une seule étape d'apprentissage (Randløv et Alstrøm, 1998; Selfridge *et al.*, 1985).

Ce que nous venons de présenter sont les points principaux de la problématique du transfert de connaissance. La suite de ce chapitre présente les principales méthodes du transfert de connaissance en apprentissage par renforcement que nous avons trouvées dans la littérature du domaine.

3.3 Méthodes actuelles de transfert

Cette section présente les principaux travaux qui ont abordé le transfert de connaissance en apprentissage par renforcement en faisant le transfert dans le processus d'apprentissage et ne pas seulement dans la conception des éléments utilisés dans l'apprentissage (représentation des états, récompense, etc.).

Parmi les travaux étudiés, nous avons pu identifier quatre groupes en fonction de la provenance de la connaissance transférée, cette dernière peut provenir :

1. De l'observation d'un autre agent. En observant un autre agent (qui peut être son mentor), l'apprenant récolte des traces qu'il peut utiliser ensuite à son bénéfice.
2. De la fonction de valeur d'une autre tâche. L'agent peut avoir résolu précédemment une tâche similaire ou une tâche plus simple que la tâche courante, disposant ainsi de la valeur Q pour le guider sur la nouvelle tâche.
3. D'une politique. Comme pour le point 2 cette politique peut avoir été apprise sur une tâche similaire ou une tâche plus simple. Par ailleurs, elle peut provenir d'une heuristique conçue pour un expert.
4. De récompenses façonnées. L'agent reçoit des récompenses additionnelles localisées (les récompenses façonnées) qui vont encourager un comportement choisi. Ces récompenses façonnées vont guider le processus d'apprentissage.

Pour chacun de ces quatre cas, il peut arriver que la connaissance disponible ne soit pas applicable directement dans le nouvel espace d'état-action, et nécessite d'être adaptée en ces sens. Nous en verrons des exemples et chaque groupe est détaillé dans les sections suivantes.

3.3.1 Apprendre en observant un autre agent

L'approche qui consiste à observer un autre agent pour accélérer le processus d'apprentissage est plutôt employé dans le domaine de la robotique (Atkeson et Schaal, 1997; Bakker et Kuniyoshi, 1996; Bentivegna et Atkeson, 2002; Bentivegna *et al.*, 2002; Schaal, 1997).

Cette approche est couramment qualifiée d'apprentissage par imitation ou d'apprentissage par démonstration. En robotique elle est composé de trois processus fondamentaux : observer les actions, les représenter et les reproduire (Bakker et Kuniyoshi, 1996). Chaque processus a sa problématique avec des questions importantes qui sont encore ouvertes.

Par exemple, il existe le problème du *mapping* des actions observées vers les actions internes de l'agent. C'est-à-dire que l'agent doit identifier les actions du mentor, se mettre à sa place et traduire chaque action vers une action qu'il peut exécuter.

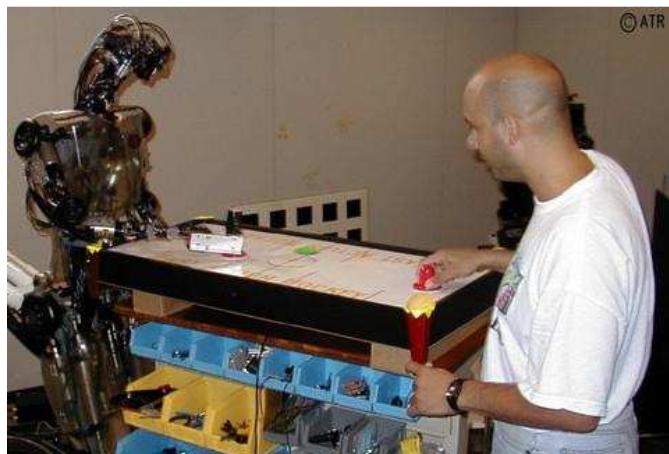


Figure 3.3 – Le robot humanoïde DB apprend à jouer en observant son adversaire (Bentivegna et Atkeson, 2002).

Dans l'apprentissage par imitation, on dispose d'une suite d'états/actions provenue de l'observation du comportement d'un autre agent. Cet ensemble peut être vu comme une politique sur une petite portion de l'espace. L'agent apprenant peut garder ses observations pour les réutiliser dans une autre situation similaire en faisant des adaptations si nécessaire ou essayer de généraliser pour les appliquer à une partie plus grande de l'espace d'états/actions ou à sa totalité.

Il existe une autre approche pour l'apprentissage à partir de l'observation d'un autre agent qui a été appelée d'imitation implicite (Price et Boutilier, 1999; Price et Boutilier, 2000; Price et Boutilier, 2003). Cette approche n'utilise pas un mentor pour guider l'exploration de l'agent apprenant ou une communication explicite entre le mentor et l'agent apprenant.

L’agent apprenant observe les transitions des états réalisées par les actions du mentor et les utilise pour mettre à jour sa fonction valeur. Dans ce cas, le rôle de mentor n’est pas explicité et les agents peuvent apprendre simplement en regardant les autres.

Une autre forme d’imitation mais avec un objectif différent est l’apprentissage par renforcement inversé¹ (Abbeel et Ng, 2004; Ng et Russell, 2000) qui consiste à extraire une fonction de récompense étant donné l’observation du comportement optimal. Imaginant comme exemple la tâche de « conduire bien une voiture » : l’approche de l’apprentissage par renforcement inversé propose d’utiliser l’observation du comportement de l’expert pour apprendre la fonction de récompense et utiliser cette dernière pour apprendre la politique au lieu d’essayer d’apprendre directement la politique avec le comportement observé. La proposition de cette approche est que la récompense est une meilleure description du comportement de l’expert et qu’elle est plus facilement utilisable dans tout l’espace d’état et actions de la tâche.

3.3.2 Transfert des valeurs (ou des Q -valeurs)

Dans certains cas, on a à disposition la fonction Q d’une autre tâche. Il existe des techniques de transfert qui « copient » la *politique de transfert*² vers la politique de l’agent apprenant en faisant le transfert des valeurs ou des Q -valeurs disponibles vers la fonction d’utilité de l’agent pour qu’il commence son processus d’apprentissage avec la fonction d’utilité initialisée avec la connaissance disponible.

Un exemple de ce type de technique est le transfert de comportement (*behavior transfer*) de (Taylor et Stone, 2005). Le transfert de comportement peut être vu comme un cas plus général de la technique d’apprentissage à partir d’une tâche plus simple, car il réutilise une fonction d’utilité qui peut provenir d’une tâche différente. Faire le transfert à partir d’une tâche plus simple consiste à augmenter progressivement la complexité de la tâche vers le niveau souhaité (Randløv et Alstrøm, 1998; Selfridge *et al.*, 1985), de manière à que l’agent puisse apprendre plus facilement des versions simplifiées de la même tâche et réutiliser cette connaissance pour augmenter la vitesse d’apprentissage pour les cas plus complexes.

Dans le cas où les espaces d’état-action des deux tâches diffèrent, il faut transformer la fonction d’utilité pour qu’elle puisse être directement appliquée aux nouveaux espaces d’états et d’actions. Une fonction de transfert de comportement ρ va permettre d’appliquer une

¹ Traduit de l’anglais : *Inverse Reinforcement Learning*

² Nous appelons politique de transfert la politique connue qui est utilisée pour le processus de transfert de connaissance.

politique de transfert à la nouvelle tâche. La fonction de transformation de la politique, ρ , va modifier la fonction d'utilité associée de manière à ce qu'elle accepte les états de la nouvelle tâche comme entrée et les actions de la nouvelle tâche comme sortie, comme illustré par la Figure 3.4. En général une politique choisit les actions pour accumuler la plus grande espérance de gain ; le problème de transformation la politique entre deux tâches se réduit à la transformation de la fonction d'utilité.

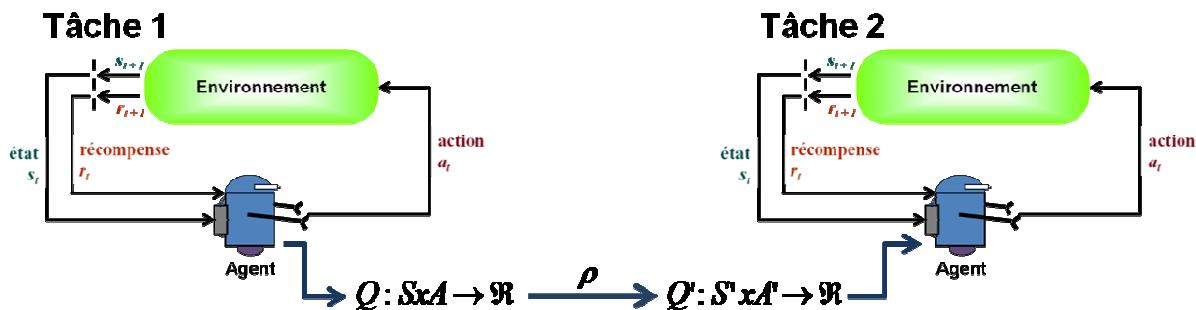


Figure 3.4 – ρ est une fonction qui transforme la fonction d'utilité Q d'une tâche pour qu'elle soit applicable dans une autre tâche qui a un espace d'états et d'actions différent.

Cette idée de transformer la fonction d'utilité et de l'insérer dans l'apprentissage de la nouvelle tâche a été testée avec le *Keepaway*¹ (Stone et Sutton, 2001). *Keepaway* est un sous-problème de la *RoboCup Soccer*² (1998; Kitano *et al.*, 1995), où une équipe doit maintenir la possession du ballon pendant que l'autre équipe essaie d'attraper le ballon ou de lui faire sortir d'une zone stipulée, qui est plus petite que le terrain entier.

(Taylor et Stone, 2005) ont créé une fonction ρ qui fait la transformation de la politique apprise pour le cas du *keepaway* de 3 vs. 2 vers le *keepaway* de 4 vs. 3. Ils utilisent des CMAC pour garder la politique et la fonction ρ fait la copie des poids du CMAC d'une politique vers l'autre avec les modifications indiquées par les auteurs.

Il existe d'autres travaux (Liu et Stone, 2006; Taylor *et al.*, 2006; Taylor *et al.*, 2007; Taylor *et al.*, 2007) qui montrent aussi qu'il est possible d'apprendre la fonction ρ pour faire le transfert de connaissance et accélérer l'apprentissage avec des techniques d'approximation de fonctions différentes, comme par exemple avec des réseaux de neurones et des réseaux bayésiens.

Les techniques de transfert des valeurs peuvent augmenter la vitesse d'apprentissage mais elles sont très dépendantes de la bonne qualité de la connaissance disponible. Et comme

¹ <http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/>

² <http://sserver.sourceforge.net/>

conséquence ces techniques de transfert sont très sensibles aux effets de transfert négatif, le résultat peut être donc catastrophique si la connaissance disponible n'est pas adapté à la tâche courante.

3.3.3 Réutilisation de politique

Lorsque la fonction Q d'une tâche corrélée n'est pas disponible, mais que l'on dispose d'une politique (ou heuristique), on ne peut plus utiliser le transfert de valeurs. Il faut donc utiliser la politique (ou heuristique) disponible d'une autre façon. En l'introduisant, par exemple, dans le processus d'apprentissage.

Un exemple de ce type d'approche est l'algorithme de réutilisation stochastique de politique (Fernández et Veloso, 2006), où l'agent apprenant fait face à trois choix : l'exploitation de la politique qu'il est en train d'apprendre, l'exploration en exécutant une action au hasard, et l'exploitation de la politique de transfert.

Cette stratégie de réutilisation de politique (appelé de π -reuse par les auteurs) est une stratégie d'exploration qui est capable d'orienter le processus apprentissage en utilisant une politique de transfert. Ils utilisent une méthode directe d'apprentissage par renforcement de manière à apprendre la fonction Q^* . A partir du moment où un algorithme d'apprentissage par renforcement peut se faire *off-policy* (c.-à-d., qu'il peut apprendre une politique en suivant une autre) (Watkins et Dayan, 1992), il peut être utilisé pour apprendre la fonction d'utilité optimale.

Dans cette technique de transfert il n'y a pas de copie des valeurs (ou Q -valeurs). La politique de transfert est employée de façon stochastique : à chaque pas de temps l'algorithme choisit aléatoirement de se fier à sa propre méthode de sélection d'action (avec une probabilité φ) ou à la politique de transfert (avec une probabilité $1 - \varphi$). Cela est exprimé par l'équation (3.1), où la politique de transfert est représentée par $\Pi_{past}(s)$ et l'exploitation de la nouvelle politique est représentée par $\Pi_{new}(s)$.

$$a = \begin{cases} \Pi_{past}(s) & \text{avec probabilité } \varphi \\ \varepsilon\text{-gloutonne}(\Pi_{new}(s)) & \text{avec probabilité } (1-\varphi) \end{cases} \quad (3.1)$$

Le transfert π -reuse suit la politique de transfert avec une probabilité φ et exploite la nouvelle politique avec une probabilité $1 - \varphi$. Comme l'exploration aléatoire est toujours demandée, il exploite la nouvelle politique en utilisant la stratégie ε -gloutonne.

De façon plus détaillée, la Figure 3.5 montre une procédure qui décrit la stratégie π -reuse avec l'algorithme *Q-Learning* intégré. La procédure reçoit comme entrées la politique de transfert Π_{past} , le nombre d'épisodes K , le nombre maximal de pas d'un épisode H , et le

paramètre φ ; et revoie la fonction Q , la politique apprise Π_{new} et le gain moyen obtenu pendant l'exécution W .

π-reuse(Π_{past}, K, H, φ, v)
1 : Initialiser $Q(s, a) = 0, \forall s \in S, a \in A$
2 : for $k = 0$ to $K - 1$ do
3 : Initialise l'état initial, s , aléatoirement
4 : $\varphi_1 \leftarrow \varphi$
5 : for $h = 1$ to H do
6 : Avec une probabilité φ_h , $a = \Pi_{past}(s)$
7 : Avec une probabilité $1 - \varphi_h$, $a = \varepsilon$ -gloutonne($\arg \max_a Q(s, a)$)
8 : Recevoir le prochain état s' , et la récompense $r_{k,h}$
9 : Mettre à jour $Q(s, a)$, et par conséquence Π_{new} :
10 : $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_a Q(s', a')]$
11 : $\varphi_{h+1} \leftarrow \varphi_h \cdot v$
12 : $s \leftarrow s'$
13 : end for
14 : end for
15 : $W = \frac{1}{K} \sum_{k=0}^K \sum_{h=0}^H \gamma^h r_{k,h}$
16 : return W , $Q(s, a)$ et Π_{new}

Figure 3.5 - Algorithme de la stratégie π -reuse

Le transfert π -reuse a été expérimenté dans un *gridworld* 24x21 (Fernández et Veloso, 2006) où l'agent démarre un épisode dans une position aléatoire de l'environnement. Chaque épisode finit soit quand l'agent atteint la sortie ou soit quand il exécute le nombre maximal de pas H . L'objectif de l'agent est de maximiser la moyenne de l'espérance de gain par épisode W , comme défini par l'équation (3.2) :

$$W = \frac{1}{K} \sum_{k=0}^K \sum_{h=0}^H \gamma^h r_{k,h} \quad (3.2)$$

Où γ ($0 \leq \gamma \leq 1$) réduit l'importance des récompenses futures, et $r_{k,h}$ définit la récompense immédiate obtenue au pas h de l'épisode k pour les K épisodes.

Le π -reuse utilise le paramètre φ qui détermine le niveau de la réutilisation de la politique de transfert. Comme la politique de l'agent évolue, ce paramètre doit changer au fil du temps si on veut assurer le meilleur niveau de transfert. Cependant, l'algorithme π -reuse le varie selon une fonction qui est fixe. Autrement dit, il va se comporter toujours de la même façon pour tout le processus d'apprentissage.

Cette approche n'assure pas la réussite du transfert qui peut échouer (et même faire du transfert négatif) si le paramètre φ n'est pas bien ajusté. De plus, trouver la fonction qui va

bien régler φ selon le changement de la politique de l'agent apprenant n'est pas du tout simple.

3.3.4 Les récompenses façonnées et les estimateurs de progrès

D'une certaine façon, tout le travail de conception et de définition de la mécanique de l'apprentissage automatique peut être considéré comme un transfert de connaissance. Dans ce cas, la connaissance d'un expert du problème est utilisé pour optimiser le plus possible le processus d'apprentissage (représentation, choix des paramètres, fonction de la récompense). Et l'utilisation de récompenses façonnées est une méthode répandue pour accélérer l'apprentissage par renforcement, l'objectif de cette méthode est de guider l'exploration vers objectif à travers des récompenses intermédiaires.

(Colombetti, 1996; Dorigo et Colombetti, 1994; Dorigo et Colombetti, 1997) sont de bons exemples de l'utilisation des récompenses façonnées. Ils ont travaillé sur l'apprentissage avec de vrais robots. Ils ont utilisé l'apprentissage par renforcement pour traduire les suggestions d'un entraîneur externe. L'entraîneur utilisé est un programme qui a un haut niveau de représentation du comportement souhaité et qui fournit une récompense façonnée.

Par exemple dans "The Hamster Experiment" (Colombetti, 1996), le robot doit apprendre à collecter des morceaux de nourriture (canettes de couleur) et à les porter à son nid. L'entraîneur fournit à l'agent une récompense en fonction de sa distance à la nourriture. Plus précisément, cette récompense est inversement proportionnelle à la distance entre le robot et les morceaux de nourriture. L'apprentissage de l'agent est essentiellement une traduction de ce qui est représenté à haut niveau par l'entraîneur vers un programme qui fait le contrôle du déplacement du robot à bas niveau.

Cette méthode de façonnage a un certain nombre d'avantages, mais aussi des inconvénients. L'agent n'a pas besoin de résoudre le problème de renforcement retardé. Mais d'un autre côté, le concepteur de l'entraîneur doit savoir à l'avance quel comportement de haut niveau est souhaité. Il doit également pouvoir juger si une simple action s'inscrit dans l'apprentissage souhaité.

Une autre manière de façonner la récompense de l'agent est d'ajouter des estimateurs de progrès (Mataric, 1994; Mataric, 1997). Cette théorie a été également testée avec les robots qui déplacent des cannettes vers leur nid. Ce façonnement incorpore la connaissance du domaine en utilisant deux types de récompenses : les récompenses hétérogènes et les estimateurs de progrès.

Les récompenses hétérogènes combinent des renforcements multimodaux d'origine externe (par exemple, sensorielle) ou interne (par exemple, états). Cette combinaison est exprimée sous la forme d'une somme pondérée des contributions propres à chaque événement. Avec cette approche chaque mode devient un sous-objectif dont l'atteinte se traduit par un signal de renforcement. Le principe de cette approche est donc l'insertion de ces sous-objectifs qui va engendrer des récompenses plus fréquentes et cela va mener l'apprentissage.

Les estimateurs de progrès sont associés à des objectifs bien précis et fixés de manière à fournir des indications sur l'évolution de l'apprentissage. Ils peuvent être vus comme des critiques internes partielles, par opposition à la critique externe (Whitehead, 1991).

Cette technique a comme avantages : la réduction de la sensibilité de l'agent aux bruits de l'environnement ; l'encouragement de l'exploration ; et la réduction de la probabilité des récompenses fortuites (c.-à-d., récompenses données à un comportement inapproprié qui par hasard a atteint un objectif).

Ces méthodes contribuent à l'évolution du processus d'apprentissage en mettant de la connaissance d'un expert du problème dans l'algorithme d'apprentissage. Cependant, cela implique plus de travail au moment de la conception de l'agent apprenant et ne fait pas du transfert de connaissance entre des tâches apprises.

Il existe des travaux qui traitent ce problème en essayant d'automatiser la construction du façonnage. Cette automatisation s'appuie sur une représentation abstraite du problème qui permet de réutiliser l'apprentissage sur d'autres problèmes avec cette même représentation. Cependant, ils ont l'inconvénient d'avoir deux processus d'apprentissage différents. Cela veut dire : plus temps pour avoir les résultats, car il faut passer plus de temps pour apprendre ; et plus de travail pour paramétrier les deux processus d'apprentissage.

Par exemple, (Konidaris et Barto, 2006) propose que les agents qui sont censés de résoudre plusieurs fois le même type de tâche sont en mesure d'apprendre leur propre façonnage, et ainsi résoudre plus vite des tâches plus difficiles en s'appuient sur un ensemble de tâches plus simples.

Cela est fait avec un apprentissage sur deux représentations distinctes, chacune dans un espace différent : une représentation dans l'espace de problème qui est Markovienne pour la tâche à accomplir ; et une représentation dans l'espace de l'agent qui n'est pas forcément Markovienne mais qui est conservée à travers les différentes tâches (dont chacune peut avoir besoin d'un nouvel espace de problème). Les agents apprennent initialement à estimer les

récompenses pour les nouveaux états à partir des « perceptions » de l'espace de l'agent pour augmenter la vitesse d'apprentissage dans l'espace du problème.

Il existe d'autres travaux qui ont approche similaire, par exemple (Bhaskara, 2007) qui fait une abstraction du PDM. Cependant dans cette approche les valeurs numériques sont apprises dans le PDM abstrait à la place du problème réel. Et cela demande plus d'exemples mais les tâches peuvent être un peu plus différent les unes des autres.

Chapitre 4

Optimisation du transfert de connaissance en apprentissage par renforcement

Dans ce chapitre nous allons présenter notre contribution au transfert de connaissance en apprentissage par renforcement. Nous avons créé un algorithme de réutilisation de politique générique et robuste étant donné qu'il est indépendant de la forme que la connaissance de transfert est disponible et qu'il s'adapte à la qualité de cette connaissance de transfert. Notre objectif avec le transfert adaptatif est d'éviter la perdre du temps avec les mauvaises politiques et de tirer partie au mieux de la connaissance, ce que ne fait pas les autres méthodes de transfert de connaissance.

A fin d'élaborer notre algorithme et de pouvoir le comparer aux travaux de transfert de connaissance existants, nous avons choisi le *gridworld* comme environnement pour les premiers essais et nous comparons notre algorithme au *Q-learning* simple et à deux autres algorithmes de transfert de connaissance.

Le reste chapitre est organisé de la façon suivante : en premier lieu nous allons présenter la plateforme d'expérimentation que nous avons conçu ; en suite nous présentons les limitations des méthodes actuelles ; pour enfin présenter notre algorithme générique de transfert de connaissance et les résultats obtenus.

4.1 Plateforme d'essai et d'expérimentation

Pour les premiers essais, évaluations et expérimentations des idées présentes dans ce travail de recherche nous avons choisi le problème de parcours du *gridworld*.

Le gridworld utilisé pour ce travail est un environnement épisodique, non déterministe, stationnaire et discret (Russell et Norvig, 1995) qui est composé d'un certain nombre de cases disposées sous une grille $n \times m$. Chaque case est un état de l'environnement où il est possible d'exécuter les actions, d'aller vers le : *nord*, *sud*, *est* ou *ouest*. Le résultat de ces actions est bruité selon une probabilité uniforme de 0,10. C'est-à-dire que le résultat d'une action exécutée sera en moyenne aléatoire une fois sur dix. Les agents ne peuvent pas traverser les murs, ni sortir de la grille.

Initialement, l'agent est positionné dans une case de la grille choisie au hasard. Il a pour objectif de sortir de la grille, c'est-à-dire, d'atteindre une case objective. L'agent a une récompense immédiate de 1 quand il arrive à la sortie et une récompense de 0 pour les autres actions. La Figure 4.1(b) montre un exemple d'un gridworld.

Nous avons choisi ce problème comme environnement d'essai parce qu'il est bien connu et utilisé dans le milieu académique (Sutton, 1996) (Boyan et Moore, 1995) (Fernández et Veloso, 2006) (Precup et Sutton, 1998) (Dietterich, 1998). De plus, il offre un bon niveau de complexité pour les premiers essais d'une méthode d'apprentissage, tout en restant relativement simple à implémenter. Notre algorithme sera expérimenté ensuite sur une tâche réelle : la simulation de situations de jeu.

Notre simulateur gridworld a eu comme base la version du « *Reinforcement Learning Repository* » de l'Université de Massachusetts (Mahadevan, 1997). Il est implémenté en C¹ et utilise des librairies X11² pour l'interface graphique. Plusieurs améliorations et modifications ont été faites, notamment en rapport avec notre approche pour le transfert de connaissance.

¹ Le langage de programmation C

² X Window System (<http://www.x.org>)

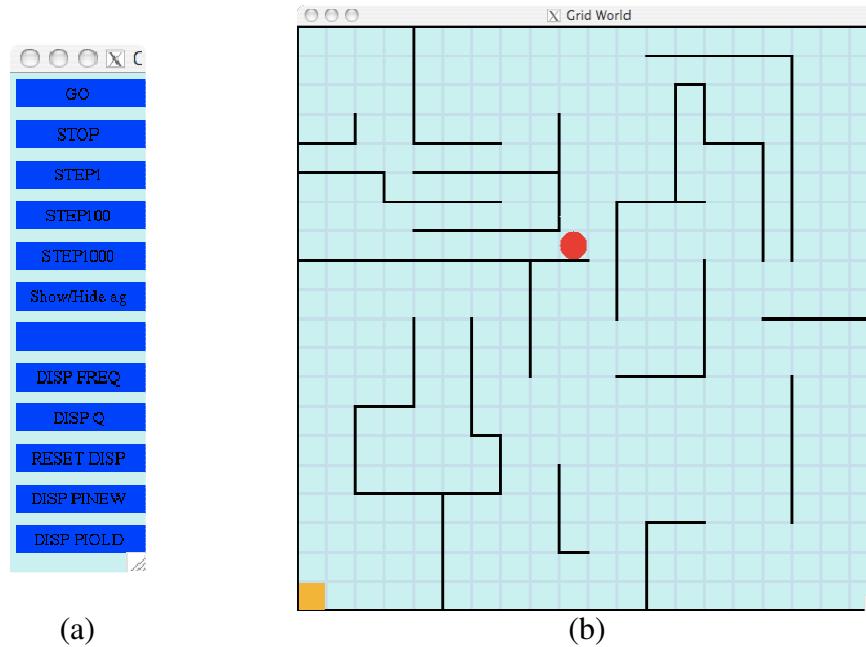


Figure 4.1 - Simulateur gridworld : (a) fenêtre de contrôle ; (b) fenêtre principale - l'agent est représenté par le cercle rouge, l'objectif par le carré jaune et les murs par les traits noirs.

4.2 Les politiques k -altérées

Afin d’expérimenter et évaluer nos algorithmes de transfert, nous avons besoin de générer des politiques plus ou moins corrélées avec cette tâche et mesurer à quel point ces politiques accélèrent l’apprentissage lorsqu’elles sont utilisées en transfert.

Il nous faut donc un moyen de générer des politiques plus ou moins corrélées avec la tâche. Pour ce faire, nous partons de la politique optimale et la dégraderons en contrôlant le taux de dégradation. Pour cela nous définissons la notion de politique k -altérée :

Définition 4.1. Soit un PDM M . Une politique déterministe π est dite k -altérée si et seulement s’il existe une politique *optimale* π^* telle que pour exactement k états, on a $\pi(s) \neq \pi^*(s)$. Pour les autres $|S| - k$ états, $\pi(s) = \pi^*(s)$.

Par exemple, une politique $|S|$ -altérée est une politique qui n’est pas en accord avec la politique optimale pour tous les états de S . Et une politique 0-altérée est une politique optimale.

Pour réaliser des expérimentations, nous avons conçu un algorithme pour générer les politiques k -altérées, pour différentes valeurs de k . Cet algorithme calcule d’abord la politique optimale, puis en perturbe k états en tirant aléatoirement la nouvelle action associé à chacun

de ces états. Au lieu de tirer cet ensemble de k états au hasard, notre algorithme tend à perturber des zones entières du *gridworld*.

Nous avons pris cette décision pour mieux représenter les imperfections d'une connaissance sous-optimale d'un problème réel, dans la mesure où ces imperfections se présentent couramment plutôt sous la forme de régions distribuées de façon bien précise. Pour voir plus de détails à propos de cet algorithme voir l'Annexe A. Cette approche représente bien la connaissance fournie par l'expert qui est pertinente sur certaines portions de l'espace seulement.

La Figure 4.2 montre un exemple d'une politique optimale (Figure 4.2(a)) et la même politique perturbée par l'algorithme (Figure 4.2(b)). La Figure 4.2(b) illustre la distribution en zones des états perturbés sur la carte du monde (voir le placement des étoiles sur la carte du gridworld).

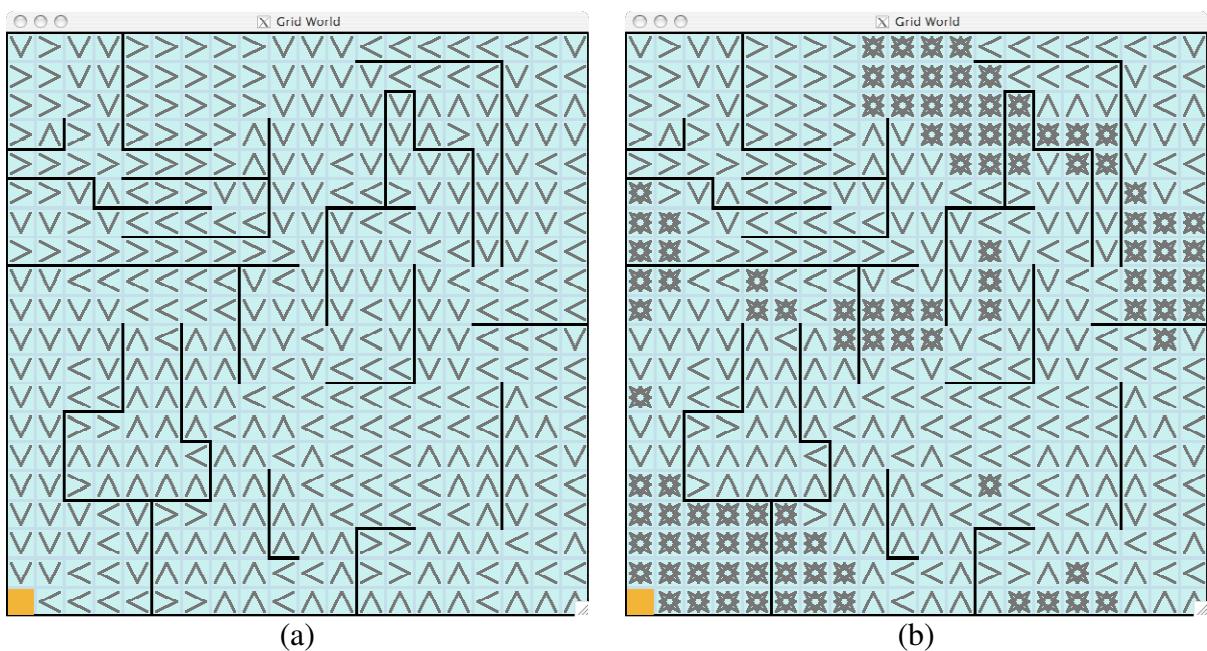


Figure 4.2 - Exemples de politiques : (a) politique optimale ; (b) politique perturbée, les états perturbés sont représentés par les étoiles au lieu de la direction indiquée par la politique

Toutes les courbes, résultats, comparaisons, etc. montrés dans le présent chapitre proviennent d'expérimentations réalisées avec le simulateur du gridworld présenté dans cette section.

4.3 Limite du transfert des politiques k -altérées

Une question que se pose est jusqu'à quel point une politique k -altérée peut être utile pour le transfert de connaissance. Soit π^* une politique optimale, et $\bar{\pi}$ une politique formée d'actions tirées aléatoirement et uniformément parmi A . Quelque soit l'état s , on a:

$$\Pr_{\bar{\pi}}(\bar{\pi}(s) = \pi^*(s)) = \frac{1}{|A|} \quad (4.1)$$

Donc, en moyenne, les politiques $\bar{\pi}$ et π^* seront d'accord sur $\frac{|S|}{|A|}$ états, et donc en désaccord sur $|S| - \frac{|S|}{|A|}$ états. Par conséquent, en moyenne, une politique aléatoire $\bar{\pi}$ sera $(|S| - \frac{|S|}{|A|})$ -altérée. On peut donc s'attendre à ce qu'une politique de transfert altérée sur plus de $(|S| - \frac{|S|}{|A|})$ états soit totalement inutile, puisque le hasard ferait aussi bien.

Prenons par exemple $|S| = 100$ et $|A| = 4$. Alors, une politique tirée au hasard sera en moyenne 75-altérée. Donc, si on dispose d'une politique 80-altérée, on ne peut pas espérer en tirer grand chose avec un algorithme de transfert.

4.4 La notion de taux de transfert

La plupart des méthodes de transfert en apprentissage par renforcement ont implicitement ou explicitement un paramètre de transfert qui détermine le niveau de la réutilisation de la politique/connaissance que l'on a à disposition.

Dans certains algorithmes de transfert, comme le π -reuse, ce paramètre apparaît explicitement. De plus, il est figé ou varie selon une fonction fixe (p. ex. une décroissance exponentielle). Dans π -reuse, ce paramètre décroît exponentiellement. Il ne dépend donc pas de la qualité de la politique de transfert ni de la dimension du problème. L'ajustement du taux de transfert est pourtant très important pour la réussite de la méthode de transfert, mais personne ne l'a jamais traité.

Dans d'autres algorithmes, comme la réutilisation de Q -valeur (Q -reuse), aucun taux de transfert n'apparaît. Néanmoins, une analyse de cet algorithme permet de mettre en évidence un tel paramètre. Pour faire ressortir un taux de transfert implicite de Q -reuse, nous allons comparer la dynamique du Q -learning simple avec celle du Q -reuse. Pour ce faire, on va simplifier ces algorithmes, en supposant qu'à chaque étape t , on est autorisé à faire *un* appel à une fonction de « *parallel sampling* » $PS(M)$ qui renvoie pour *chaque* état-action (s,a) un état suivant s' et une récompense r . Cette hypothèse courante dans l'analyse des algorithmes

d'apprentissage permet de s'affranchir de l'exploration (voir (Kearns et Singh, 1999) pour plus de détails). On suppose aussi qu'à chaque instant, les deux algorithmes obtiendront les mêmes réponses de $PS(M)$, c'est-à-dire que les valeurs de r et s' seront identiques pour un état-action-étape donné.

L'algorithme de *Q-learning* standard est donc modélisé ainsi : à chaque étape t , $PS(M)$ est appelé, et le calcul suivant est effectué :

$$Q_{t+1}(s, a) = (1 - \alpha)Q_t(s, a) + \alpha r + \alpha\gamma \max_{a'} Q_t(s', a')$$

Initialement, pour tous les états-actions, on a $Q_0(s, a) = 0$. L'algorithme de *Q-reuse* est identique au *Q-learning*, sauf que les *Q-valeurs* à l'étape 0 ne sont pas initialisées à zéro, mais reflètent la politique à transférer. On notera les *Q-valeurs* de l'algorithme du *Q-reuse* par $\bar{Q}_t(s, a)$. On peut montrer par induction que $\bar{Q}_t(s, a) \geq Q_t(s, a)$. On a donc :

$$\bar{Q}_{t+1}(s, a) = (1 - \alpha)\bar{Q}_t(s, a) + \alpha r + \alpha\gamma \max_{a'} \bar{Q}_t(s', a')$$

On montre donc le résultat suivant :

Proposition. Soient $\bar{Q}_t(s, a)$ et $Q_t(s, a)$, les *Q-valeurs* obtenues avec les algorithmes de *Q-reuse* et de *Q-learning*. Il existe une fonction $\varphi_{s,a}(t)$ vérifiant $0 \leq \varphi_{s,a}(t) \leq \|\bar{Q}_0\|_\infty (1 - \alpha + \alpha\gamma)^t$ telle que $\bar{Q}_t(s, a) = Q_t(s, a) + \varphi_{s,a}(t)$.

Proof. Soit $\Delta_t = \bar{Q}_t - Q_t$. Notons que $\Delta_0(s, a) = \bar{Q}_0(s, a)$. Développons les formules précédentes :

$$\begin{aligned} \Delta_{t+1}(s, a) &= (1 - \alpha)\Delta_t(s, a) + \alpha\gamma \left\{ \max_{a'} \bar{Q}_t(s', a') - \max_{a''} Q_t(s', a'') \right\} \\ &\leq (1 - \alpha)\Delta_t(s, a) + \alpha\gamma \max_{a'} \Delta_t(s', a') \\ &\leq (1 - \alpha + \alpha\gamma) \max_{s', a'} \Delta_t(s', a') \\ &\leq (1 - \alpha + \alpha\gamma)^{t+1} \|\Delta_0\|_\infty \end{aligned}$$

Avec $\|\Delta_0\|_\infty = \max_{s,a} \Delta_0(s, a)$. Cette dernière ligne nous permet déjà d'écrire $0 \leq \bar{Q}_t(s, a) \leq Q_t(s, a) + (1 - \alpha + \alpha\gamma)^{t+1} \|\Delta_0\|_\infty$, ce qui prouve la proposition.

Ceci nous permet de voir l'algorithme *Q-reuse* comme un algorithme de *Q-learning* avec un biais $\varphi_{s,a,t}$.

Cette proposition montre que le *Q-reuse* se comporte comme un algorithme de *Q-learning* auquel un « bonus d'exploration » $\varphi_{s,a}(t)$ guide l'exploration vers la politique de transfert. De plus, $\varphi_{s,a}(t)$ décroît exponentiellement vite, à une vitesse déterminée par α et γ . De même, que pour l'algorithme π -*reuse*, le *Q-reuse* dispose donc d'un paramètre implicite

réglant le taux de transfert. Nous montrerons par la suite que ce taux ne convient pas pour toutes les situations, et qu'il gagnerait à être optimisé.

De façon générale, pour la plupart des algorithmes de transfert, on peu identifier le paramètre φ dans l'intervalle $[0, 1]$ qui indique le pourcentage de l'utilisation de la politique de transfert. Quand $\varphi = 0$ nous n'utilisons pas la politique de transfert, nous sommes donc dans le cas de l'apprentissage par renforcement standard. En revanche, quand $\varphi = 1$ nous faisons entièrement confiance à la politique de transfert et l'agent n'utilise que la politique de transfert pour choisir ses actions.

4.5 Influence du taux de transfert sur l'apprentissage

Comme nous l'avons vu, le *Q-reuse* n'a pas de taux de transfert explicite et le « bonus d'exploration » décroît toujours de la même façon. En conséquence, l'évolution de l'apprentissage est très sensible à la qualité ou pertinence de la connaissance transférée. Ainsi, il faut être sûr de ce que l'on transfère parce que cela peut améliorer ou bien dégrader la performance de l'apprentissage (cela est illustré dans l'Annexe F).

Il y a d'autres méthodes de transfert de connaissance (Fernández et Veloso, 2006) qui essaient d'affronter le problème de la qualité de la connaissance que l'on veut transférer en faisant une pondération entre la politique/connaissance de transfert et la politique/connaissance de l'agent.

Mais normalement, au bout d'un certain temps la connaissance de l'agent apprenant évolue avec le processus d'apprentissage et devient meilleur que la politique de transfert. Et comme conséquence la pondération n'est plus valide.

Cependant, ces méthodes demandent un important travail de paramétrage pour trouver le bon taux de transfert. L'inconvénient de cette approche est qu'une fois la bonne configuration trouvée pour le transfert, elle est spécifique à la situation courante. Comme cette situation change pendant le processus d'apprentissage, soit par l'évolution du comportement de l'agent ou soit par un autre facteur quelconque, il faut aussi modifier les paramètres. Et cette modification des paramètres est nécessaire jusqu'à la convergence de l'apprentissage. Ces méthodes sont donc sensibles à la variation de la politique, et donc sensibles à la variation de k pour les politiques k -altérées.

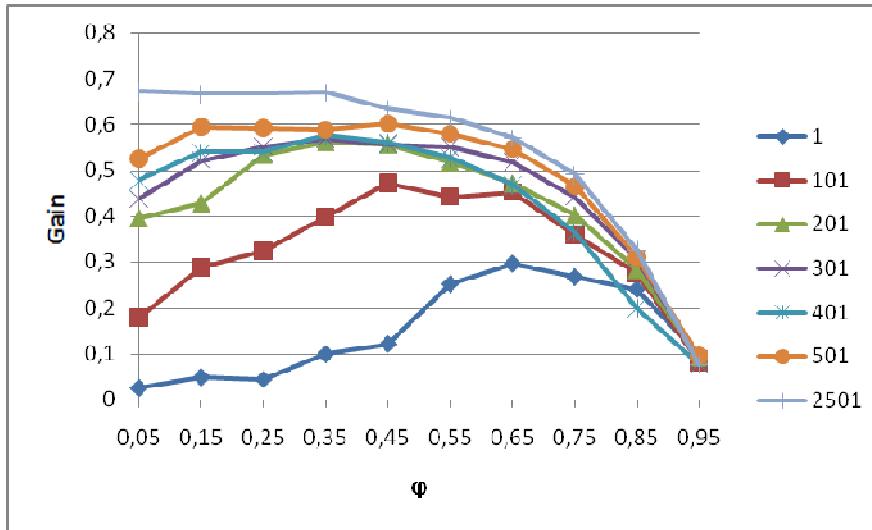


Figure 4.3 – Résultats de différents paramètres pendant l'apprentissage

La Figure 4.3 présente les différentes performances de l'algorithme de transfert π -reuse (Fernández et Veloso, 2006) en utilisant une politique de transfert *k-altérée* dans des différents étapes de l'apprentissage. Pour différentes valeurs de φ , ces courbes montrent le gain de la politique apprise au bout d'un certain nombre d'épisodes (1 épisode, 101 épisodes, 201 épisodes, etc.).

On remarque que fixer φ à zéro (pas de transfert) ou à un (transfert total : l'apprenant suit la politique *k-altérée*) ne donne pas les meilleurs résultats. Idéalement, il faudrait fixer φ à 0,65 en début d'apprentissage, puis vers l'épisode 100, fixer φ à 0,4, ce qui mènerait à un apprentissage bien plus rapide.

Ceci s'explique par deux raisons : d'abord, la politique à transférer est très altérée, et donc un φ trop élevé nuirait à l'exploration ; ensuite, plus l'agent apprend, et plus il doit exploiter, au lieu de se fier à la politique de transfert, ce qui explique pourquoi φ devrait décroître.

En regardant des courbes comme celles de la Figure 4.3, on peut imaginer un système, ou un mécanisme quelconque, qui va ajuster l'utilisation de la politique de transfert selon ses résultats. Cela est le principe de notre algorithme de transfert adaptatif.

En résumé, le transfert de valeurs n'est pas toujours la meilleure option pour le transfert parce que tout simplement il y a des cas où l'on n'a pas de fonction à copier et aussi parce qu'une simple copie des valeurs peut être très sensible à la qualité de la politique de transfert et présenter des résultats catastrophiques (comme illustré par la Figure 7.27). Les autres méthodes qui réutilisent une politique de transfert d'une manière plus élaboré demandent de grands efforts de la part du concepteur pour l'adaptation et la mise en place de la méthode.

utilisée à l'algorithme d'apprentissage par renforcement. Ils sont donc sensibles à la variation de k pour les politiques k -altérées.

Nous voulons, donc, régler le paramètre φ d'une façon automatique en fonction de toute la conjecture du problème. C'est justement avec cet objectif que nous avons créé l'algorithme d'optimisation du transfert présenté par la section suivante. Il s'agit d'une approche originale pour le traitement du taux de transfert en apprentissage par renforcement.

Ces difficultés ont motivé une des contributions du présent travail de recherche. La construction d'un algorithme de transfert générique qui optimise l'utilisation de la connaissance disponible quelle que soit la qualité de la politique de transfert. Il s'agit d'un algorithme qui va automatiquement trouver l'équilibre entre l'utilisation de la politique de transfert et la politique de l'agent, en tenant en compte les résultats obtenus par la politique de transfert et par la politique actuelle de l'agent. Cet algorithme est présenté par la section suivante.

4.6 L'algorithme de transfert adaptatif

Notre objectif est de concevoir un algorithme générique de transfert de connaissance dans lequel le taux φ s'adapte à la qualité de la politique. Par exemple, si la politique disponible n'aide pas à explorer correctement, il faut que l'algorithme fasse tendre φ vers zéro rapidement.

Pour obtenir un algorithme générique, nous proposons une méthode permettant de modifier les algorithmes de transfert dans lesquels φ apparaît explicitement (par exemple le π -reuse ou le *memory-guided exploration* (Todd et al., 2001)). Pour simplifier la présentation de cet algorithme, nous le décrivons appliqué au π -reuse, mais d'autres méthodes de transfert pourraient être utilisées.

Appelons $learn(\bar{\pi}, \varphi)$ la fonction qui déroule un épisode d'apprentissage en utilisant le π -reuse. Cette fonction reçoit comme paramètres la politique de transfert $\bar{\pi}$ et le taux de transfert φ , et renvoie le gain obtenu.

```

learn( $\bar{\pi}, \varphi$ )
1 : Initialise l'état initial,  $s$ , aléatoirement
2 : for  $h = 1$  to  $H$  do
3 :   Avec une probabilité  $\varphi$ ,  $a = \bar{\pi}(s)$ 
4 :   Avec une probabilité  $1 - \varphi$ ,  $a = \varepsilon$ -gloutonne( $\arg \max_a Q(s, a)$ )
5 :   Recevoir le prochain état  $s'$ , et la récompense  $r_h$ 
6 :   Mettre à jour  $Q(s, a)$ , et par conséquence  $\pi$  :
7 :      $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]$ 
8 :    $s \leftarrow s'$ 
9 : end for
10: return  $R = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \dots$ 

```

Figure 4.4 – La fonction $learn(\bar{\pi}, \varphi)$

L'idée de notre algorithme adaptatif est d'encapsuler cette fonction dans un algorithme qui optimise φ à chaque épisode, cela est illustré par la Figure 4.5 :

```

1 : Initialiser  $Q(s, a) = 0, \forall s \in S, a \in A$ 
2 : for  $k = 0$  to  $K - 1$  do
3 :    $\varphi \leftarrow \text{optimise\_}\varphi(\bar{\pi}, \pi)$ 
4 :    $R \leftarrow \text{learn}(\bar{\pi}, \varphi)$ 
5 : end for

```

Figure 4.5 – Algorithme de transfert adaptatif

Notre est facile à décrire, mais pour avoir le résultat souhaité, il faut trouver la bonne valeur φ . Pour cela nous avons créé la fonction $\text{optimise_}\varphi(\bar{\pi}, \pi)$ qui tente d'optimiser φ pour la politique de transfert utilisée et la politique actuelle de l'agent apprenant. L'efficacité du taux de transfert ou la qualité du comportement combiné ($\bar{\pi}$ et π) de l'agent est mesuré avec le critère d'évaluation défini pour le problème. L'important est que nous pouvons l'évaluer et par conséquent trouver la meilleure valeur de φ pour le critère choisi. La fonction $\text{optimise_}\varphi(\bar{\pi}, \pi)$ sera détaillée par la suite dans la section 4.8.

Le point critique de l'algorithme est que la politique de l'agent apprenant change au fil du temps, en obligeant le changement de la valeur de φ si l'on veut maintenir le bon équilibre. Cependant d'un point de vue pratique, nous ne pouvons pas tout réévaluer à chaque pas d'apprentissage t , parce que cette réévaluation est coûteuse au niveau du temps de calcul.

Il faut donc trouver un mécanisme de mise à jour de φ que soit en même temps efficace et pas très couteux du point de vue du temps de calcul. Nous présentons deux solutions pour faire cette mise à jour (sections 4.7 et 4.9). Ces solutions sont évaluées (section 4.10) en

comparaison avec l'algorithme d'apprentissage par renforcement (sans transfert) et avec les méthodes de transfert des valeurs et π -reuse.

4.7 Évaluation hors ligne

Le premier algorithme se base sur le principe d'une évaluation espacée et sur hypothèse de que φ ne varie pas beaucoup entre deux évaluations (hypothèse confirmée par nos expérimentations). Cet algorithme fait une évaluation tous les n épisodes.

Le changement pour réaliser l'évaluation tous les n épisodes d'apprentissage est simple, et cet algorithme est montré dans la Figure 4.6. Nous pouvons voir à la ligne 3 de l'algorithme que la valeur de φ n'est calculé que tous le n épisodes.

```

1 :   Initialiser  $Q(s,a) = 0, \forall s \in S, a \in A$ 
2 :   for  $k = 0$  to  $K - 1$  do
3 :       Tous les  $n$  épisodes :  $\varphi \leftarrow \text{optimise\_} \varphi(\bar{\pi}, \pi)$ 
4 :        $R \leftarrow \text{learn}(\bar{\pi}, \varphi)$ 
5 :   end for
```

Figure 4.6 – Transfert adaptatif et l'évaluation hors ligne

Comme nous arrêtons le processus d'apprentissage pour faire une évaluation et trouver la valeur de φ et puis continuer l'apprentissage, l'évaluation est dénommée hors ligne.

4.8 Implémentation de la fonction $\text{optimise_} \varphi(\bar{\pi}, \pi, \dots)$

Pour les premières expérimentations avec l'évaluation hors ligne montrés dans le présent travail de thèse, nous avons choisi un algorithme d'évaluation qui ne donne pas forcement la meilleure valeur de φ , mais qui donne des bonnes indications de cette valeur et qui n'est pas complexe à implémenter.

Afin d'optimiser φ , nous essayons différentes valeurs de φ dans un intervalle définit et nous choisissons celui qui s'adapte le plus au critère choisi (c.-à-d. celui qui donne le meilleur résultat). En donnant plus de détail, nous redéfinissons la fonction $\text{optimise_} \varphi(\bar{\pi}, \pi)$ par $\text{optimise_} \varphi(\bar{\pi}, \pi, k, n_e, p_{\max}, c)$, où nous ajoutons d'autres paramètres qui sont utilisés pour la méthode de calcul de φ .

L'algorithme consiste à calculer le gain moyen (équation (3.2)) pour tous les φ_i dans l'intervalle $[0 ; 1]$ tel que $\varphi_i = i/k$, où $i = 0,1,2,3,\dots,k$, en exécutant n_e épisodes avec une limite maximale de pas de p_{max} .

Choisir le paramètre φ qui produit le meilleur gain ne donne pas les meilleurs résultats. En effet, cela revient à privilégier fortement l'exploitation de la politique de transfert sur l'exploration. Par exemple, si $\varphi = 1$, avec le π -reuse, aucune exploration n'est effectuée.

Pour palier ce problème, nous introduisons un nouveau paramètre $c \in [0;1]$, qui représente la proportion de gain que l'on abandonne en profit de l'exploration. Concrètement, si $c = 10\%$, on cherche donc la plus petite valeur de φ dont le gain soit au moins 90% du meilleur gain. Le faisant, on s'assure que l'exploration persiste, tout en gardant un gain élevé.

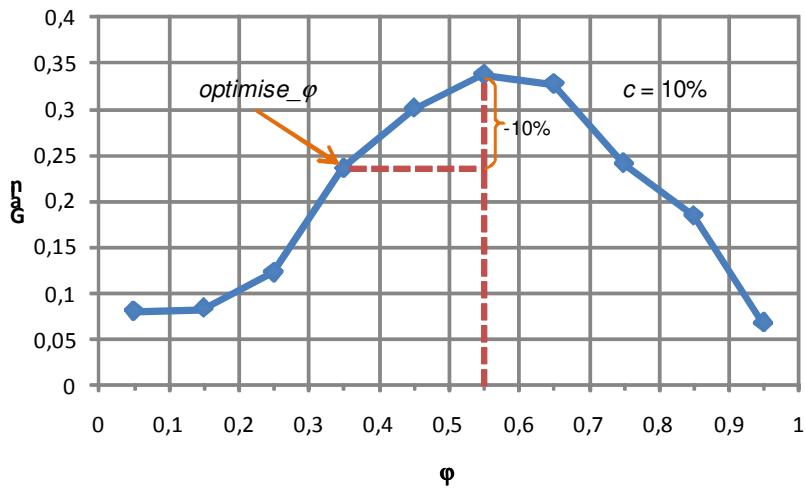


Figure 4.7 – Évaluation de φ

La Figure 4.7 montre un exemple du gain trouvé pour différents valeurs de φ . Selon les résultats de la Figure 4.7 le meilleur φ est 0,55 et il va définir la zone d'acceptation des résultats avec le paramètre c . L'algorithme va donc choisir le plus petit φ qui a un résultat dans cette zone.

Cet algorithme d'évaluation prend en compte les difficultés pour optimiser φ . Tout d'abord, on ne peut qu'estimer la performance de φ en échantillonnant sa valeur. Ensuite, les courbes des performances de φ ne sont pas toujours lisses comme la Figure 4.7. Enfin, la fonction que l'on optimise est la valeur de la politique mixte. Donc, on ne privilégie pas forcément l'exploration, mais plutôt l'exploitation. Nous avons fait de nombreux essais avec le résidu de Bellman (en donnant donc plus d'importance à l'exploration), mais cela n'a pas

donné des bons résultats parce qu'explorer tout l'espace d'état en cherchant la valeur optimale pour la fonction d'utilité n'est pas intéressant. Principalement pour les problèmes de grande taille, où il faut beaucoup de temps pour commencer à avoir de bons résultats avec cette approche. Nous cherchons à guider l'exploration afin d'avoir de bons résultats le plus tôt possible.

4.9 Évaluation en ligne

Notre deuxième algorithme de transfert repose sur une évaluation en ligne. C'est-à-dire qu'il évalue et choisit la valeur de φ sans arrêter le processus d'apprentissage. Le fait d'évaluer et choisir φ en apprenant en même temps peut permettre de gagner du temps par rapport à l'algorithme précédent parce que l'agent apprend en continu.

Néanmoins cela rend le calcul du meilleur φ plus difficile, parce que l'on évalue une politique qui n'est pas stable et on ne peut pas permettre de prendre beaucoup de temps pour faire cette évaluation. De plus, un mauvais choix de φ peut dégrader la performance de l'apprentissage.

Notre algorithme de transfert de connaissance avec l'évaluation en ligne utilise des principes de la *descente du gradient* et du *recuit simulé* (Cornuéjols et Miclet, 2002; Russell et Norvig, 1995). Où nous allons dans la direction de la plus forte pente vers le meilleur résultat avec un déplacement inspiré de la température de l'algorithme du recuit simulé. La distance de déplacement est grande au début puis elle est diminuée au fur et à mesure.

Le principe de l'algorithme est de faire varier φ dans les deux directions possibles (en donnant plus d'importance à la politique de transfert ou plus d'importance à la politique d'agent) et prendre celle qui présente le meilleur résultat. Avec l'hypothèse de que l'on peut appliquer un algorithme de gradient, on déplace φ dans la meilleure direction et on recommence la procédure avec un déplacement plus petit. Le pseudocode de l'algorithme avec l'évaluation en ligne est donné par la Figure 4.8.

Nous faisons donc n épisodes d'apprentissage avec $\varphi_1 = \varphi + \Delta$ et n épisodes avec $\varphi_2 = \varphi - \Delta$. Puis nous les résultats, celui qui obtient la meilleure valeur sera le prochain φ . Ensuite nous décroissons la valeur de Δ et nous refaisons les n épisodes avec les nouveaux valeurs.

L'évaluation consiste à faire une moyenne du gain obtenu sur les n épisodes d'apprentissage. Comme l'agent apprend à chaque épisode (c.-à-d. sa politique évolue en

continu), nous alternons les épisodes avec $\varphi_1 = \varphi + \Delta$ et $\varphi_2 = \varphi - \Delta$ afin d'avoir une comparaison plus fiable. En faisant cette alternance entre φ_1 et φ_2 , nous comparons deux politiques qui sont plus proches par rapport au temps d'apprentissage.

```

1 :   Initialiser  $n, c, \Delta, \Delta_{\min}, \varphi$ 
2 :   repeat
3 :      $\varphi_1 \leftarrow \varphi - \Delta$ 
4 :      $\varphi_2 \leftarrow \varphi + \Delta$ 
5 :      $S_1 \leftarrow S_2 \leftarrow 0$ 
6 :     repeat  $n$  fois
7 :        $S_1 \leftarrow S_1 + learn(\bar{\pi}, \varphi_1)$ 
8 :        $S_2 \leftarrow S_2 + learn(\bar{\pi}, \varphi_2)$ 
9 :     until  $n$  fois
10:    if ( $S_1 \geq S_2 (1 - c)$ ) then
11:       $\varphi \leftarrow \varphi_1$ 
12:    else
13:       $\varphi \leftarrow \varphi_2$ 
14:    if ( $\Delta \geq \Delta_{\min}$ ) then  $\Delta \leftarrow \Delta / 1.5$ 
15:  until apprentissage n'est pas finie

```

Figure 4.8 – Transfert adaptatif et l'évaluation en ligne

Ces sont les choix que nous avons fait pour les premières expérimentations de notre transfert de connaissance avec l'évaluation en ligne. Nous avons pensé à des solutions plus élaborées pour l'évaluation (voir le Chapitre 7), mais nous avons par principe préféré commencer par une solution plus simple à fin de nous laisser la possibilité de la faire évoluer par la suite.

4.10 Expérimentations et résultats

Cette section présente les expérimentations et les résultats que nous avons obtenus sur le *gridworld*, tout en faisant une comparaison avec les deux principales méthodes que nous avons trouvées dans la littérature du transfert de connaissance en apprentissage par renforcement : les méthodes de transfert des valeurs (Taylor et Stone, 2005) et la méthode π -*reuse* (Fernández et Veloso, 2006).

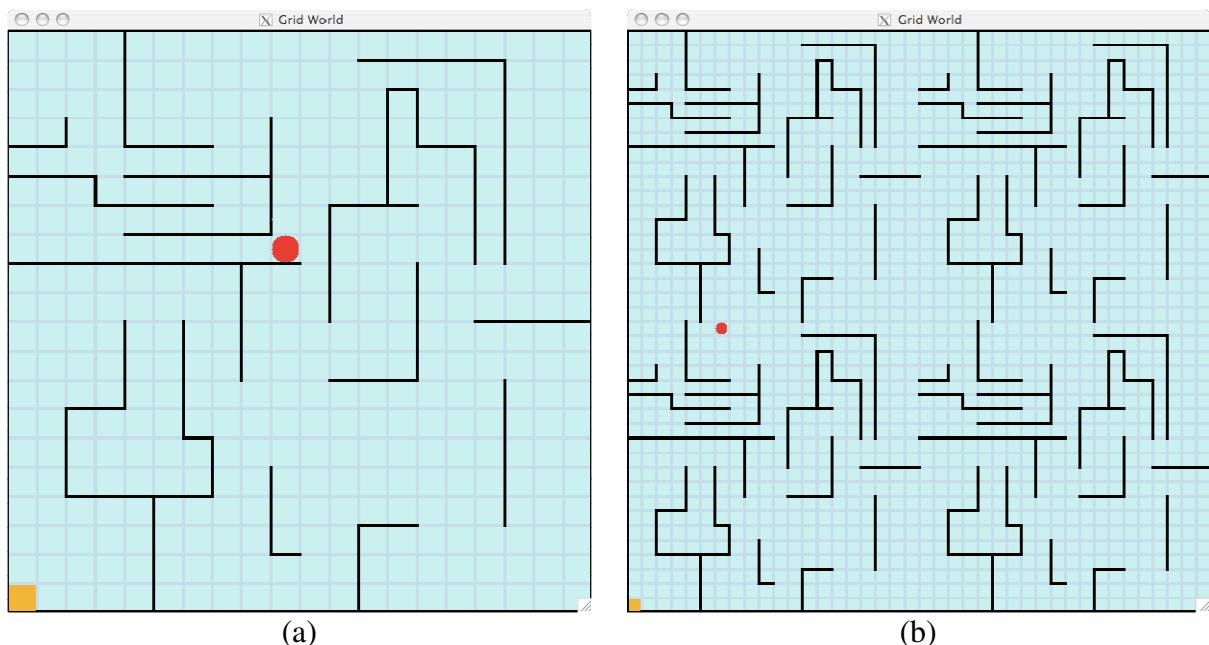
Dans un premier temps, nous exposons comment nous avons fait les expérimentations et par la suite nous présentons les résultats obtenus en même temps que nos commentaires.

4.10.1 Procédure d’expérimentation avec le *gridworld*

Nous avons déjà dit qu’une des raisons pour le choix de gridworld comme premier environnement d’essai est son utilisation dans le milieu académique, de cette façon nous pouvons nous situer plus facilement par rapport aux travaux de référence.

Pour la même raison nous avons choisi un gridworld de 20x20 cases (c’était la taille utilisée par (Fernández et Veloso, 2006)), cependant nous voudrions voir aussi les performances dans un monde plus grand et nous avons donc fait également des expérimentations avec un monde quatre fois plus grand que le premier (40x40 cases). Cela nous a permis d’évaluer la robustesse des algorithmes de transfert pour différentes tailles de l’espace d’état.

Les cartes des deux mondes utilisés sont montrées par la Figure 4.9. Ces deux cartes ont permis à la fois de nous mieux comparer (c.-à-d. avec les mêmes dimensions couramment utilisées) aux travaux déjà existants dans la littérature du transfert en apprentissage par renforcement et de voir les résultats dans un monde un peu plus grand.



**Figure 4.9 – Les deux cartes utilisées : (a) gridworld 20x20 soit 400 cases ;
(b) gridworld 40x40 soit 1600 cases**

Avec les deux cartes définies nous avons fait tourner l’algorithme du *Q-Learning* de base pour trouver deux politiques optimales pour les mondes choisis. Ces deux politiques optimales sont utilisées pour créer nos politiques de transfert qui seront *k*-altérées (voir Définition 4.1). Les politiques optimales trouvées sont présentées par la Figure 4.10.

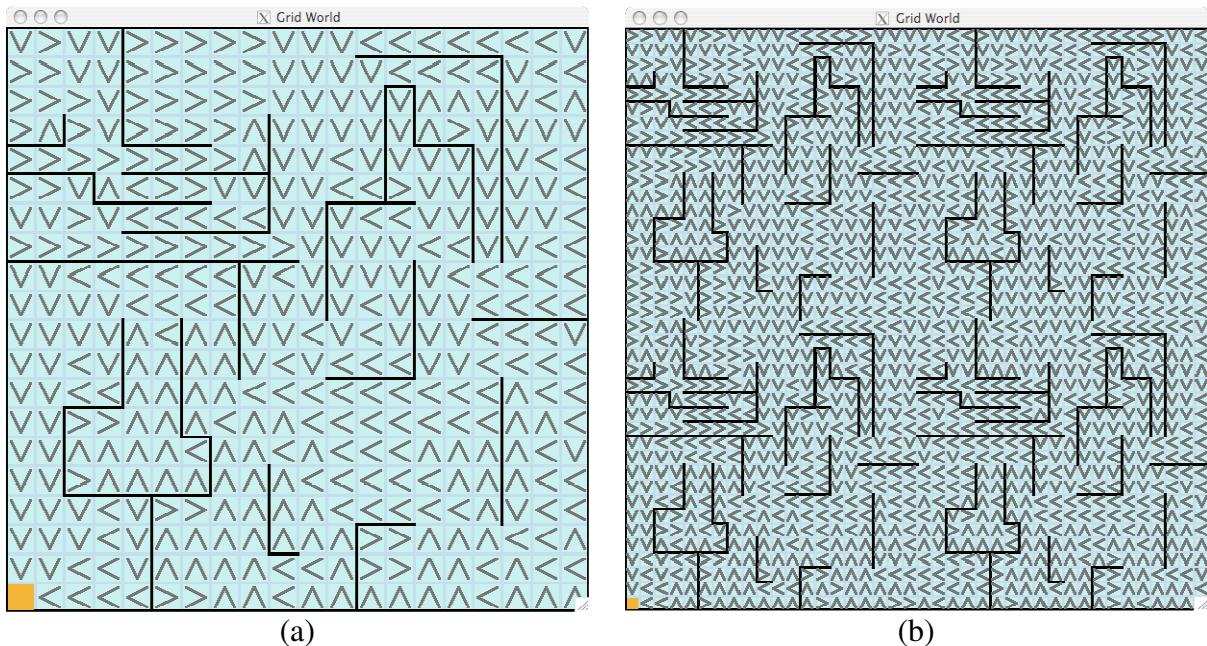


Figure 4.10 – Les politiques optimales pour les deux mondes d’expérimentation :
(a) gridworld 20x20 ; (b) gridworld 40x40

Pour créer les politiques de transfert *k*-altérées (Définition 4.1), nous avons utilisé l’algorithme de l’Annexe A. Cet algorithme représente bien les imperfections des heuristiques sous-optimales sur un problème réel, comme expliqué précédemment. Par exemple, il est plus fréquent que l’on ne connaisse pas un quartier entier d’une ville qu’une liste de rues tirées au hasard.

Nous avons crée des politiques de transfert avec 10%, 25% et 50% des états perturbées. Cela veut dire des politiques 40-altérées, 100-altérées et 200-altérées pour le gridworld de 20x20 et des politiques 160-altérées, 400-altérées et 800-altérées pour le gridworld de 40x40.

Comme notre objectif est de montrer les résultats de notre approche et de comparer ces résultats avec les travaux que nous avons trouvés dans la littérature du transfert de connaissance, nous avons altéré la politique optimale (cf. section 4.2) pour créer les politiques de transfert.

Les politiques générées sont convenables pour les méthodes de transfert *Q-reuse* et *π -reuse*, elles ne font donc pas du transfert négatif. Cela veut dire que nous confrontons nos résultats dans un cadre bien adapté aux techniques de (Fernández et Veloso, 2006; Taylor et Stone, 2005), car ces techniques n'ont pas un mécanisme pour éviter les mauvaises politiques de transfert.

Nous avons utilisé trois politiques *k-altérées* différentes pour chaque pourcentage d'états perturbés, elles ont été choisies dans le but d'avoir des perturbations dans différentes zones de la carte du *gridworld*. Les politiques utilisées pour le *gridworld* 20x20 sont présentées par : Figure 4.11, Figure 4.12 et Figure 4.13. Nous avons choisi d'utiliser ces politiques parce les connaissances des différents régions de la carte ont un impact différent sur le gain de l'agent. Il y a des régions que l'agent passe plus souvent ou qui vont amener plus facilement à l'objectif.

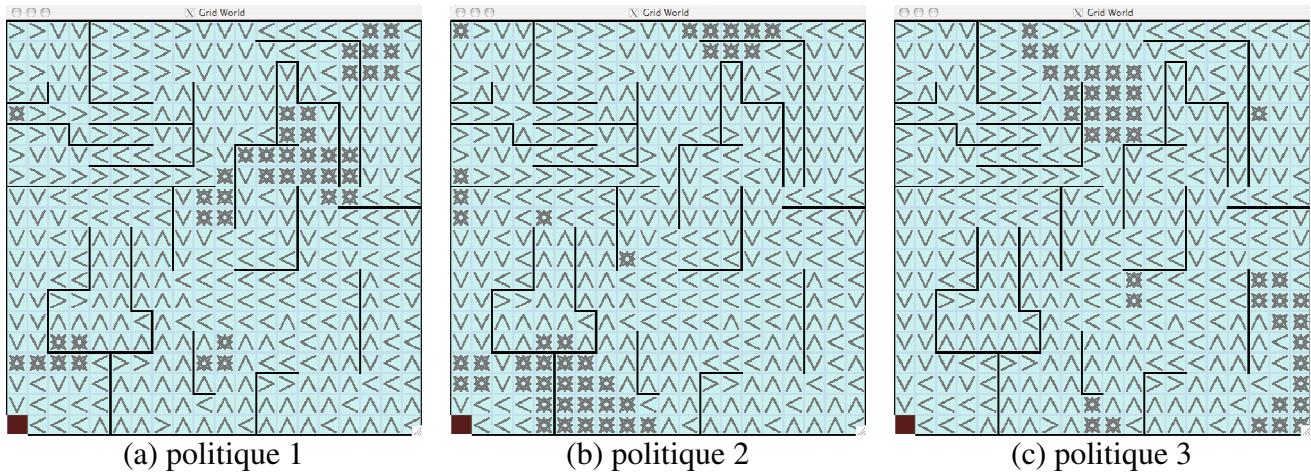


Figure 4.11 – Les politiques 40-altérées (10% de perturbation - gridworld 20x20)

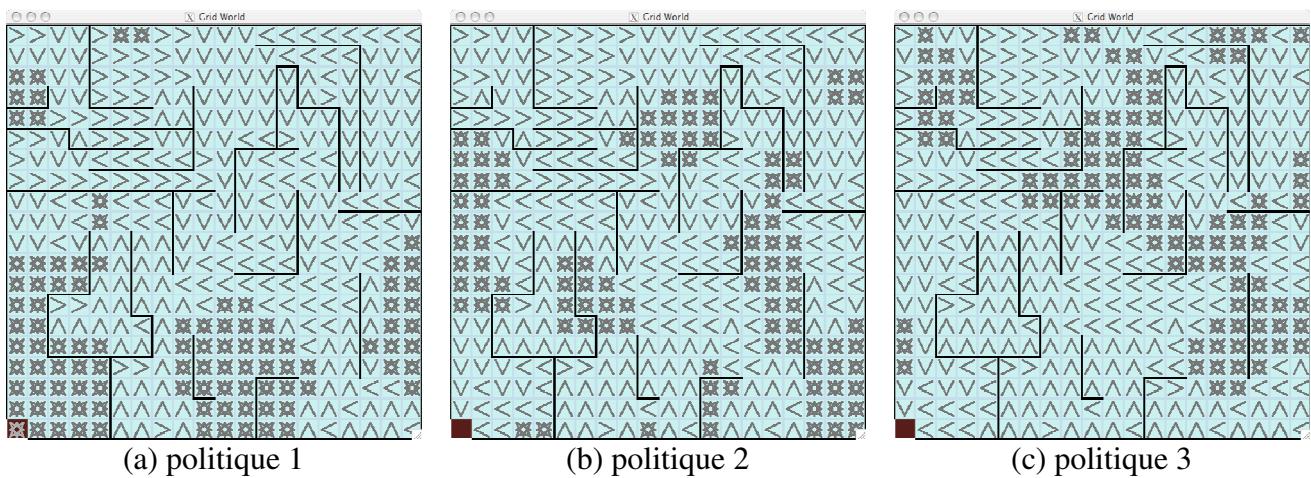


Figure 4.12 – Les politiques 100-altérées (25% de perturbation - gridworld 20x20)

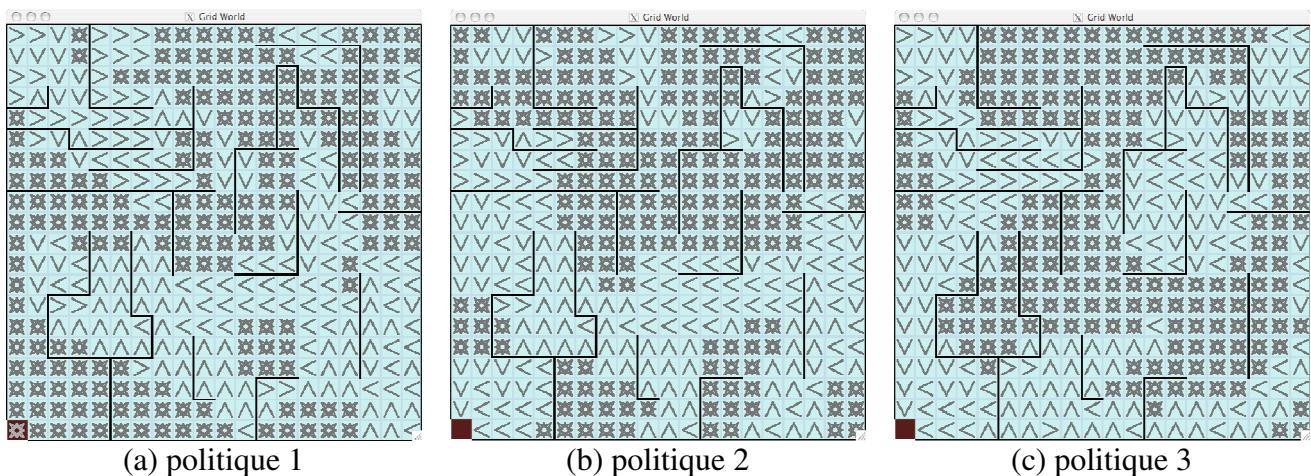


Figure 4.13 – Les politiques 200-altérées (50% de perturbation - gridworld 20x20)

En suivant le même principe de trois politiques pour chaque pourcentage d'états perturbées nous avons la Figure 4.14, la Figure 4.15 et la Figure 4.16 pour le *gridworld* de 40x40.

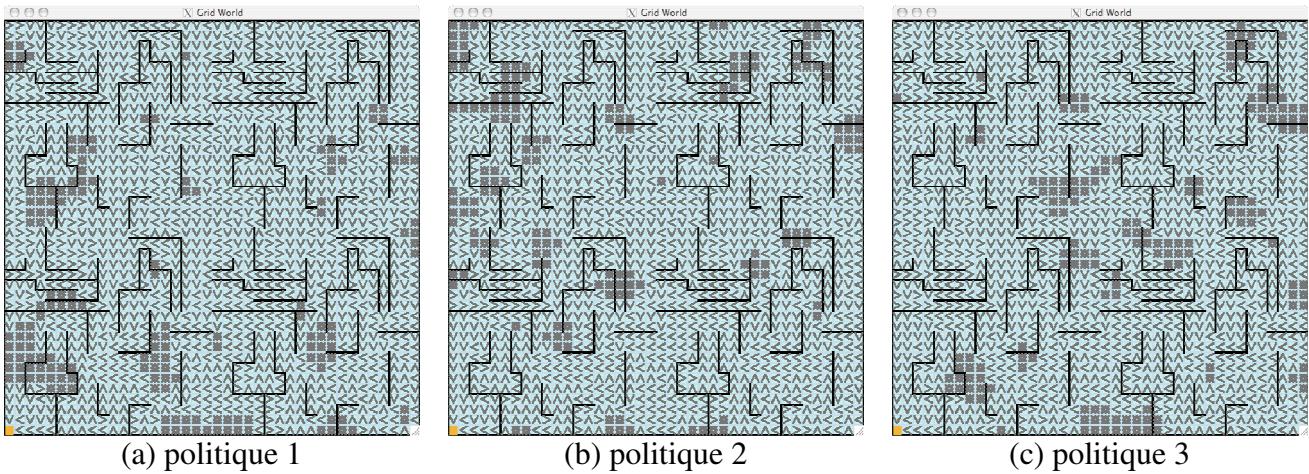


Figure 4.14 – Les politiques 160-altérées (10% de perturbation - gridworld 40x40)

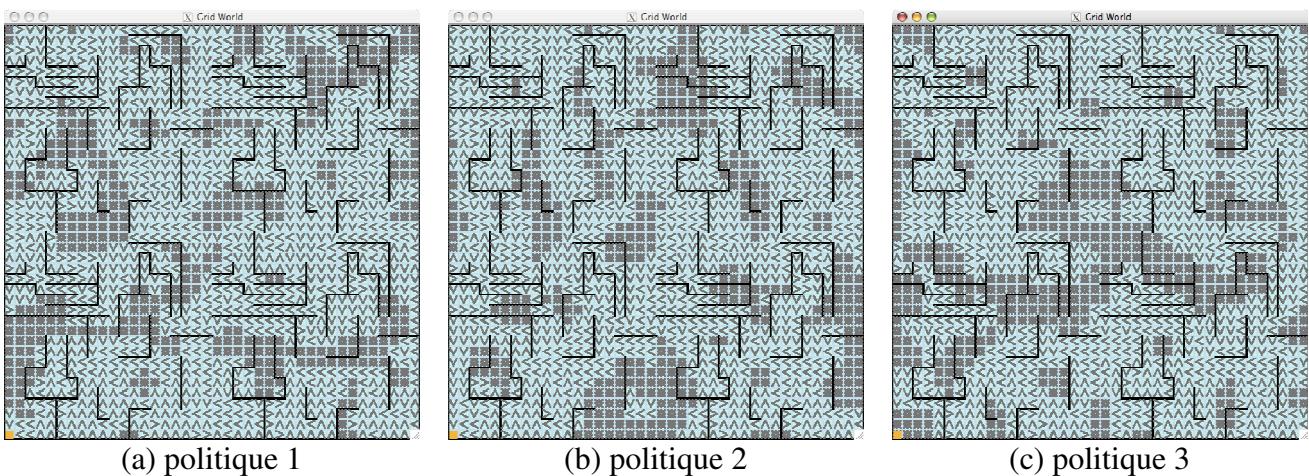


Figure 4.15 – Les politiques 400-altérées (25% de perturbation - gridworld 40x40)

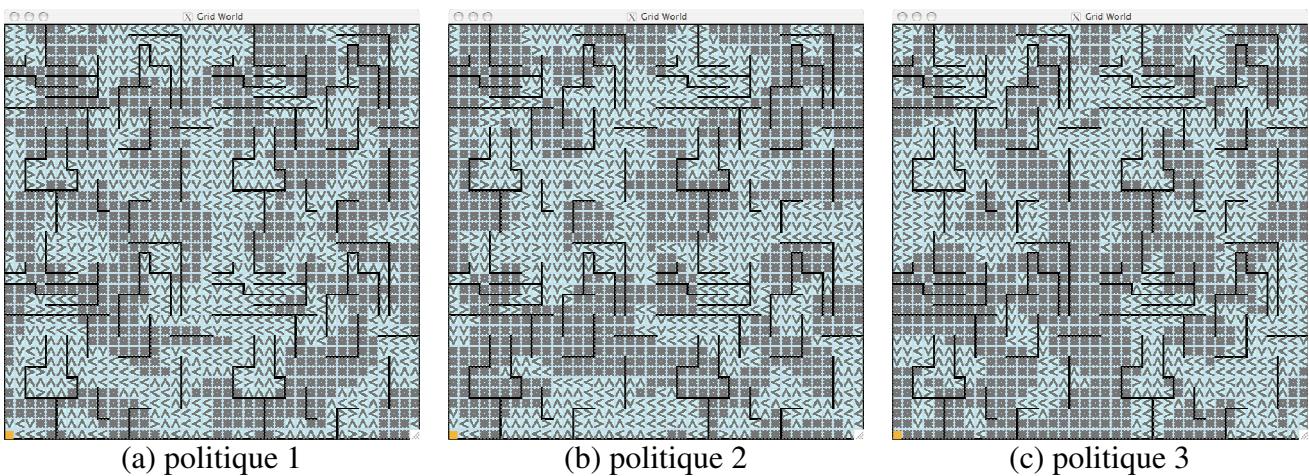


Figure 4.16 – Les politiques 800-altérées (50% de perturbation - gridworld 40x40)

Pour des raisons de mise en page la taille de ces figures a été réduite et, à première vue, le symbole qui indique une case perturbée peut ressembler à un carré, mais il s'agit du même symbole utilisé précédemment.

4.10.2 Résultats avec le *gridworld*

Afin de générer les résultats de chaque configuration nous avons fait 30 expérimentations de 8000 épisodes avec une longueur maximale de 200 pas. L'apprentissage par renforcement a été fait en utilisant le *Q-Learning* avec les paramètres : $\gamma = 0,99$, un $\alpha = 0,10$ et un $\varepsilon = 0,95$. Ces paramètres se sont montrés empiriquement bien adaptées au domaine d'application choisi. L'algorithme d'optimisation du transfert avec l'évaluation hors ligne a fait une évaluation (composée de 300 épisodes) tous les 50 épisodes avec $k = 10$.

L'application de chaque algorithme produit une courbe qui montre la moyenne du gain obtenu (axe des ordonnées) sur les 30 expérimentations en fonction du déroulement des épisodes d'apprentissage (axe des abscisses). Dans le but d'améliorer la lisibilité des résultats, les courbes de cette section présentent les résultats des premiers 4000 épisodes, cela est la partie la plus importante pour le transfert de connaissance.

Nous avons créé trois figures pour chaque expérimentation, la première sert à comparer les différents algorithmes d'apprentissage par rapport au résultat obtenu. La deuxième sert à comparer les algorithmes par rapport au temps d'exécution de chaque expérimentation. Et la troisième figure, qui fait un gros plan sur le premier groupe de courbes de la deuxième figure. Cette dernière sert pour voir principalement la différence entre la méthode hors ligne et les autres méthodes, bien que la performance de la méthode en ligne par rapport aux autres méthodes.

Dans toutes les courbes de gain, nous avons les épisodes sur l'axe des abscisses, et sur l'axe des ordonnées nous avons la moyenne du gain obtenu des 30 expérimentations. Et les méthodes que nous avons déjà exposées dans le présent travail de thèse sont représentés par les légendes : *Q-Learning* ; Transfert des valeurs (transfert de connaissance avec copie de la fonction valeur) ; π -reuse (réutilisation de la politique) ; Evaluation hors ligne (transfert avec évaluation hors ligne) et Evaluation en ligne (transfert avec évaluation en ligne).

Dans toutes les courbes de temps d'exécution, nous avons les expérimentations sur l'axe des abscisses, et sur l'axe des ordonnées nous avons le temps en secondes pour réaliser l'expérimentation en utilisant toujours la même machine. Pour des raisons de visualisation nous présentons deux ensembles de courbes. Le premier ensemble présente toutes les courbes, dans ce premier ensemble de courbes nous pouvons voir la différence entre la méthode d'évaluation hors ligne et les autres méthodes. Le deuxième fait un gros plan sur les courbes qui sont groupés plus proche de l'axe, c.-à-d. toutes les méthodes à l'exception de la méthode hors ligne. De cette façon nous pouvons bien voir toutes les courbes et la distance entre eux.

Le premier ensemble de courbes est montré par la Figure 4.17, cet ensemble de courbes correspond aux expérimentations avec le gridworld 20x20 et 10% des états perturbés. Dans ce cas, les deux propositions que nous avons faites (transfert avec évaluation en ligne et hors ligne) ont eu des très bons résultats par rapport au *Q-Learning* de base et ils ont été meilleur que les deux autres méthodes de transfert. Sauf pour la politique 3 où le transfert de valeurs a été plus adaptée.

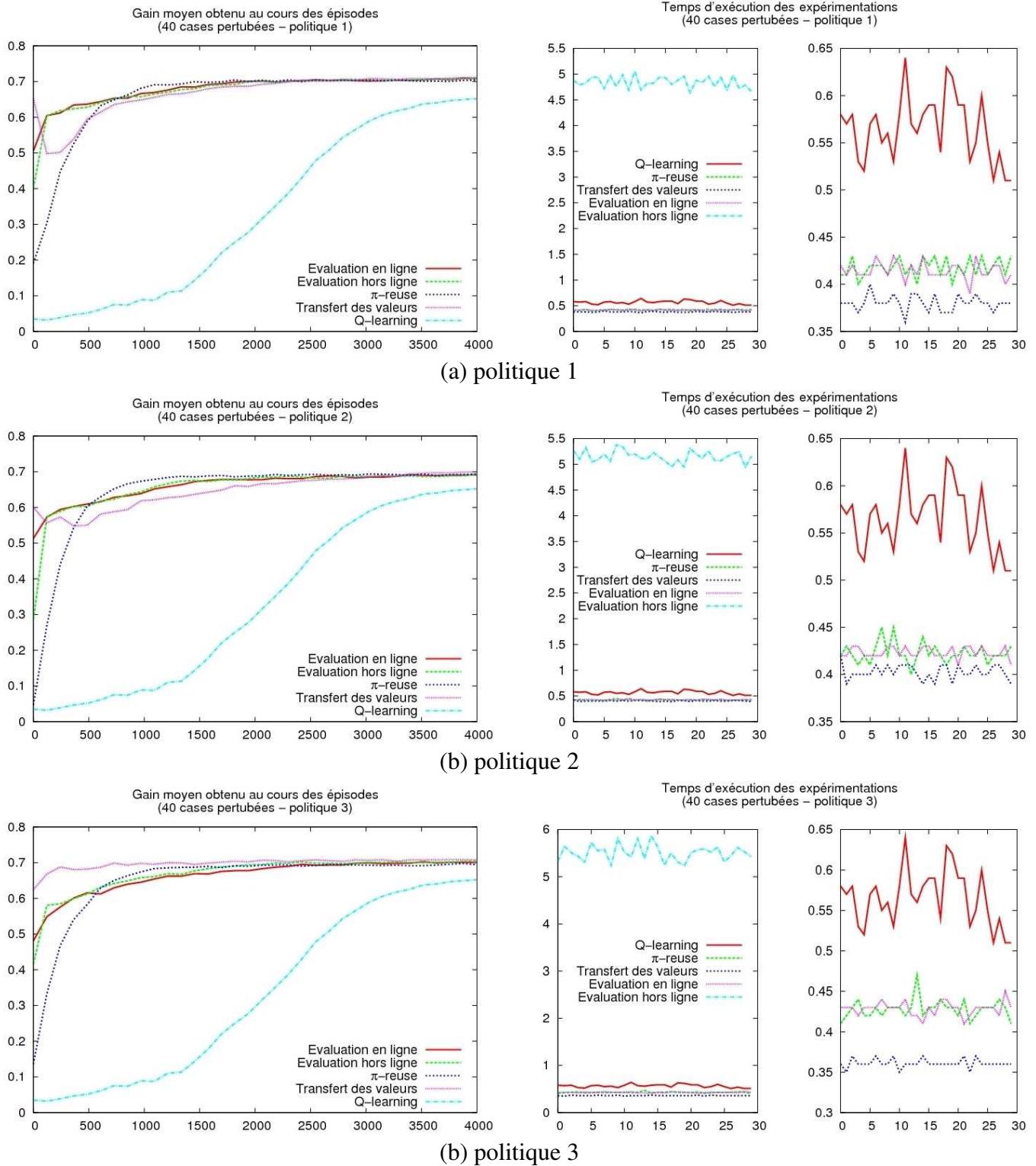


Figure 4.17 – Les résultats pour les politiques 40-altérées (10% de perturbation)
gridworld 20x20

Le deuxième ensemble de courbes (Figure 4.18) montre les résultats pour les politiques avec 25% des états perturbés utilisés pour le gridworld 20x20. Pour la première politique la méthode de transfert de valeurs a donné lieu à une performance inférieure aux autres

méthodes de transfert, dans lesquelles les deux méthodes avec l'évaluation (hors ligne et en ligne) ont eu un petit avantage par rapport au π -reuse.

Dans la politique 2 nous ne pouvons pas affirmer quelle méthode a été la meilleur car il y a une succession de changement de position. A l'exception de l'évaluation hors ligne qui a maintenu une bonne régularité pendant toute l'expérimentation.

Pour la dernière perturbation de cet ensemble (Figure 4.18(c)) le transfert de valeurs a développé une bonne performance. Le transfert de valeurs a été suivie des deux méthodes de transfert avec évaluation (hors ligne et en ligne) et de la méthode π -reuse qui a amélioré l'évolution de ses résultats après un certain temps d'apprentissage.

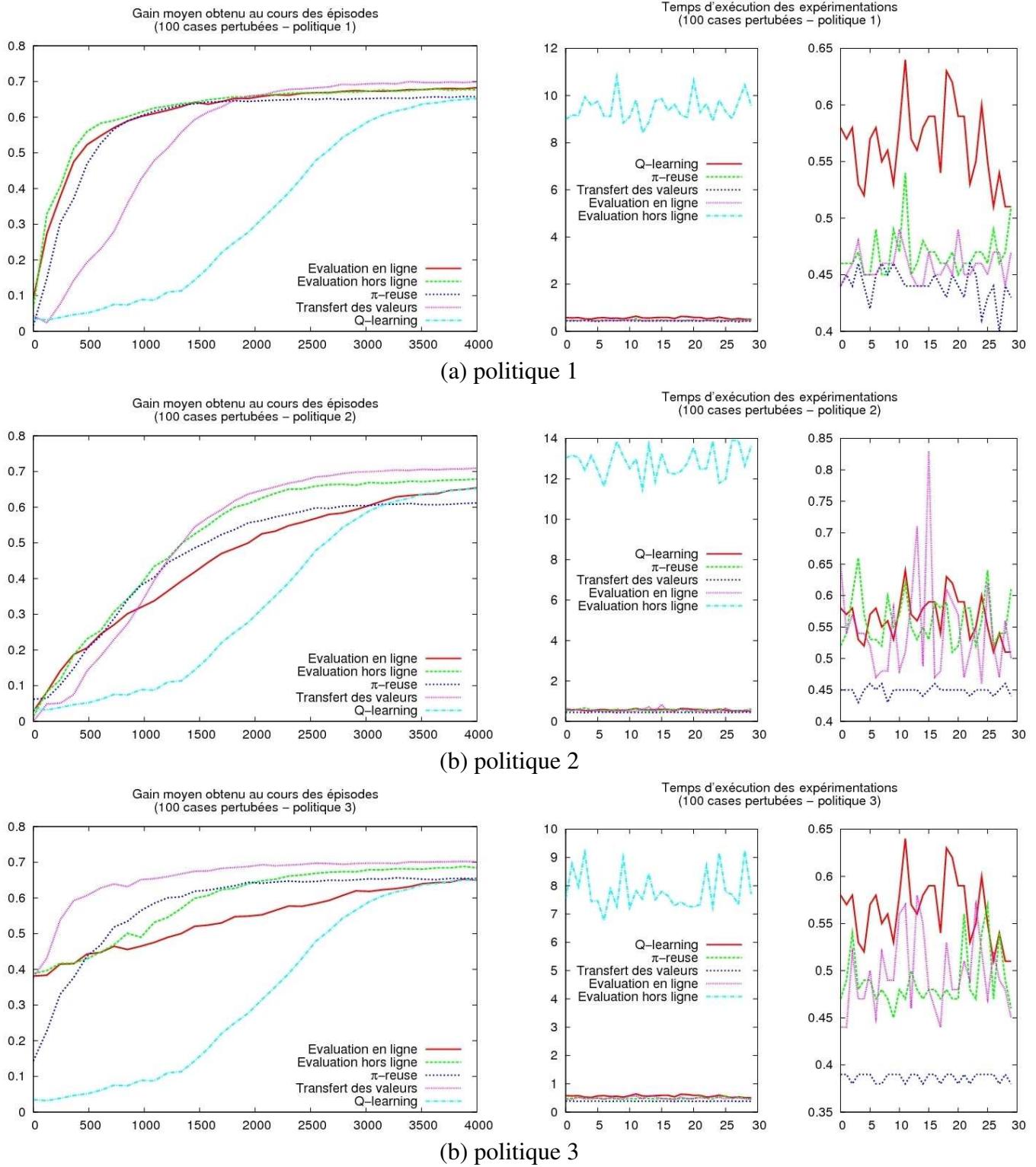


Figure 4.18 – Les résultats pour les politiques 100-altérées (25% de perturbation)
gridworld 20x20

En regardant maintenant les résultats du transfert avec l'utilisation des politiques avec 50% des états perturbés (Figure 4.19) nous pouvons voir pour la politique 1 (Figure 4.19(a)) que les méthodes de transfert marchent relativement bien avec un avantage pour l'évaluation

hors linge. Suivie de l'évaluation en ligne et du π -*reuse* qui sont à peu près au même niveau, l'évaluation en ligne ne va être meilleure que vers l'épisode 2200.

Les résultats de la deuxième politique (Figure 4.19 (b)) montrent une bonne évolution du gain obtenu par les méthodes d'évaluation hors ligne, d'évaluation en ligne et de π -*reuse*. Le transfert de valeurs commence comme le *Q-Learning* mais montre une meilleure évolution après un certain nombre d'interactions avec l'environnement.

Cependant le transfert de valeurs évolue différemment avec la politique 3 (Figure 4.19 (c)) en montrant la meilleure performance. Elle est suivie de l'évaluation hors ligne, puis de l'évaluation en ligne. La méthode π -*reuse* viens tout de suite après avec un résultat très proche de l'évaluation en ligne, mais un peu inférieur à court et à long terme.

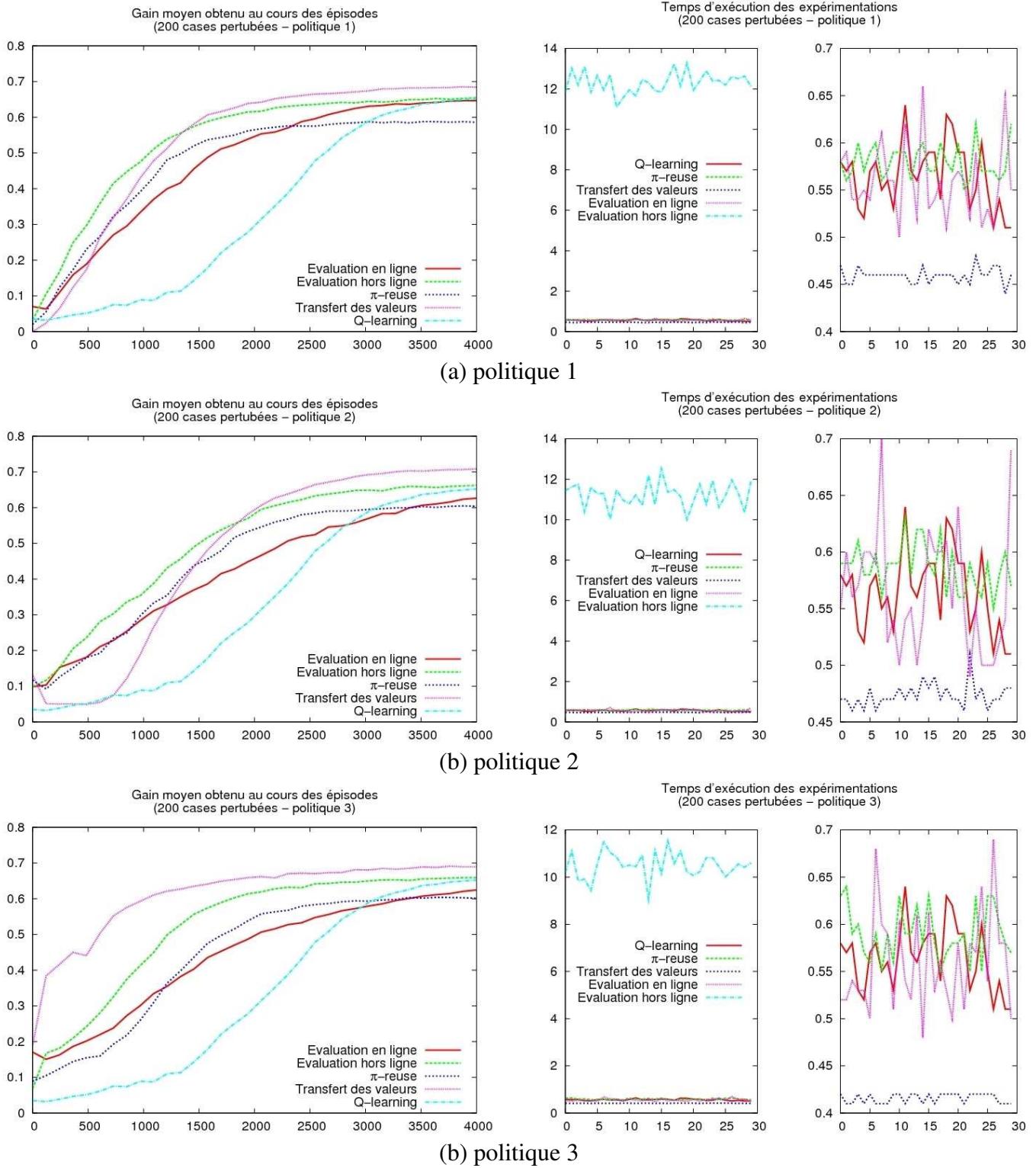


Figure 4.19 – Les résultats pour les politiques 200-altérées (50% de perturbation)
gridworld 20x20

Les autres courbes ci-après montrent les résultats des algorithmes testés avec le gridworld 40x40 (Figure 4.9(b)) et les perturbations de la Figure 4.14, de la Figure 4.15 et de la Figure 4.16.

En regardant les courbes du gain obtenu avec les politiques qui ont 10% des états perturbés (Figure 4.17), nous pouvons voir que les algorithmes de transfert de connaissance avec l'évaluation hors ligne et en linge ont obtenus les meilleurs résultats. A l'exception de la politique 2 (Figure 4.14(b)) qui a favorisé l'algorithme de transfert des valeurs.

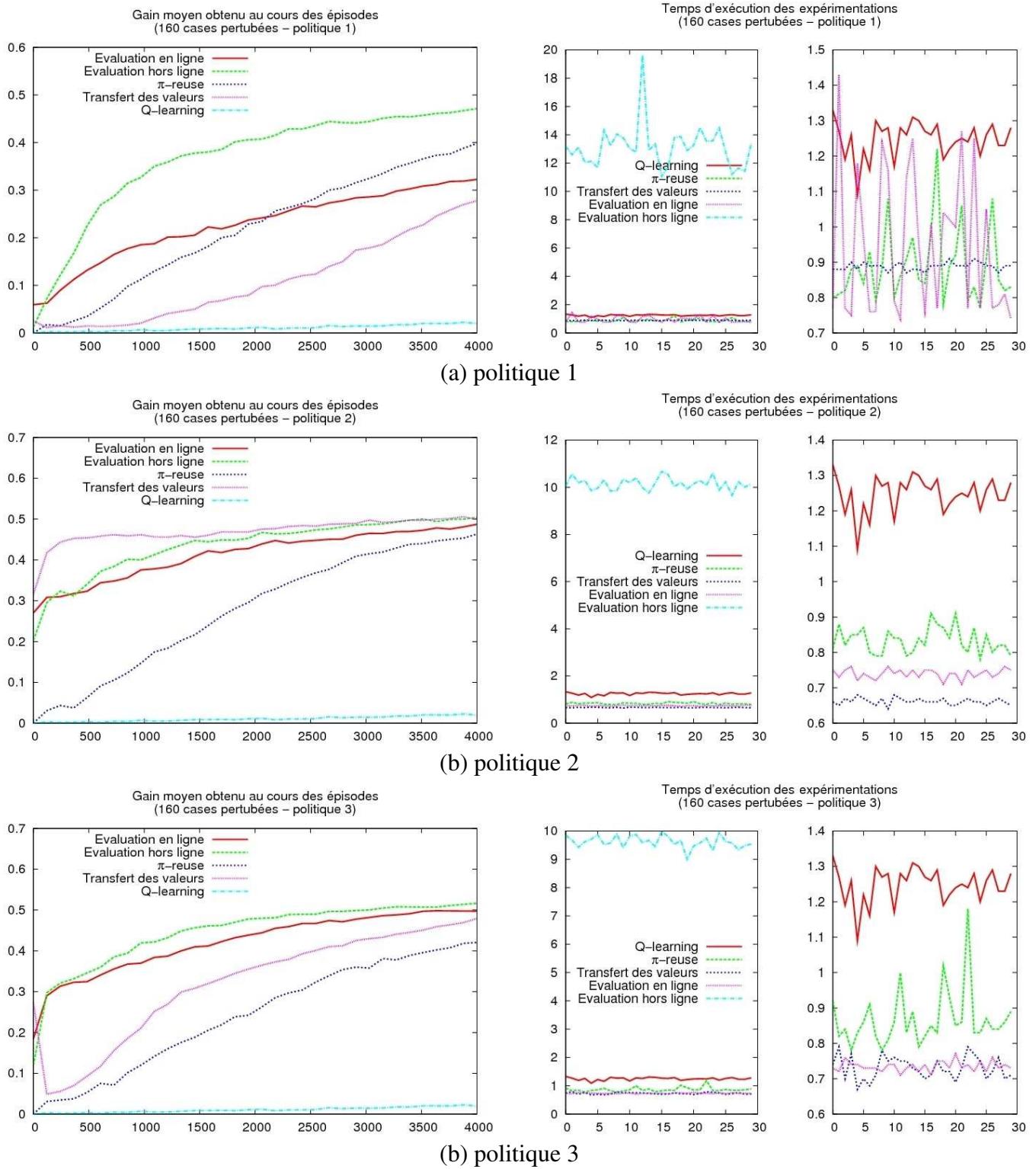


Figure 4.20 – Les résultats pour les politiques 160-altérées (10% de perturbation)
gridworld 40x40

La Figure 4.21 montre que la méthode de transfert de valeurs a eu des résultats proches du *Q-Learning* de base, sauf pour la politique 2 où il arrive aux mêmes résultats que les méthodes avec l'évaluation (hors ligne et en ligne).

En regardant les courbes nous pouvons dire, sans aucun doute, que la méthode de transfert avec l'évaluation hors ligne a eu les meilleurs résultats. Par rapport aux autres méthodes nous pouvons signaler que l'évaluation en ligne a pris la deuxième place, car elle a eu à peu près les mêmes résultats que le π -*reuse*, cependant elle a été meilleur pour la politique 2.

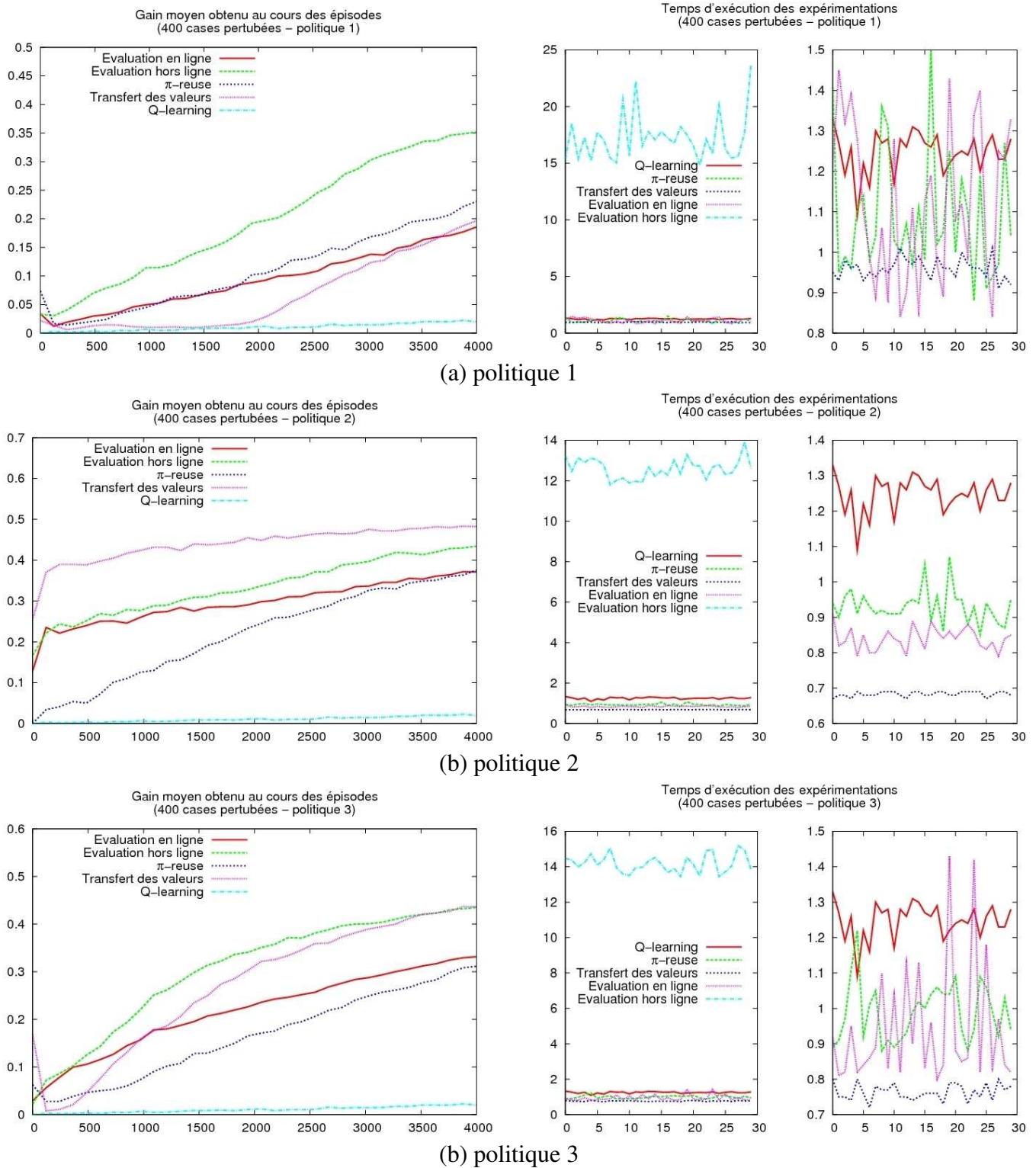
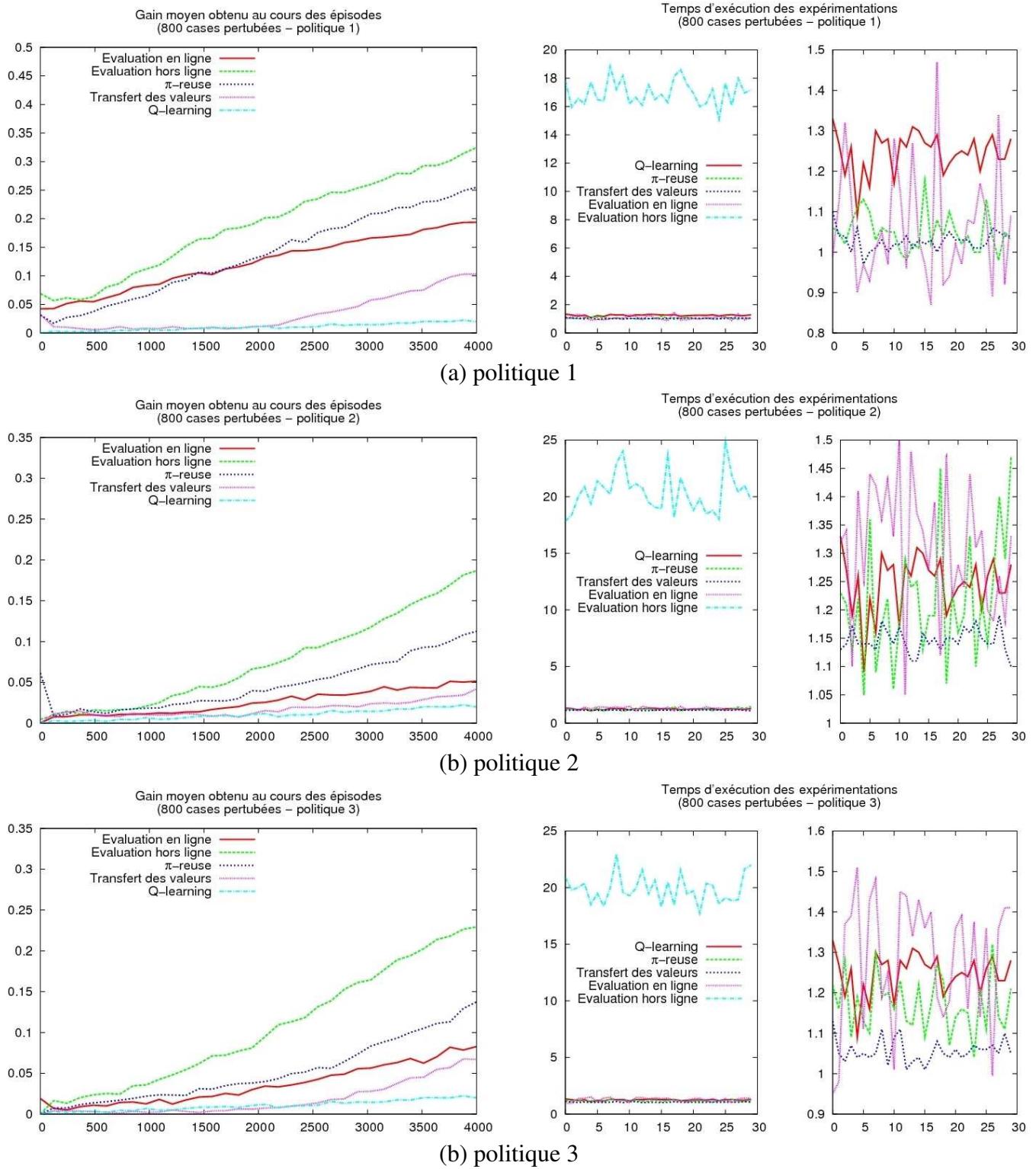


Figure 4.21 – Les résultats pour les politiques 400-altérées (25% de perturbation)
gridworld 40x40

Les courbes de la Figure 4.22 montrent que les méthodes avec l'évaluation hors ligne et en ligne s'adaptent bien même pour un transfert avec 50% des états perturbés.

L'évaluation hors ligne prend la première place pour toutes les politiques. En même temps nous pouvons signaler que l'évaluation en ligne est en moyenne meilleure que les autres méthodes de transfert, bien que cette méthode soit dépassée par le transfert de valeurs pour la politique 3.

Même si le transfert de valeurs est un peu meilleur que le π -reuse pour la politique 3, nous pouvons considérer qu'elles sont à peu près au même niveau de gain.



**Figure 4.22 – Les résultats pour les politiques 800-altérées (50% de perturbation)
gridworld 40x40**

4.11 Discussions

Nos courbes montrent en premiers lieu que l'utilisation d'une méthode de transfert de connaissance dans le processus d'apprentissage par renforcement a des avantages par rapport au processus d'apprentissage par renforcement de base.

Cependant il faut signaler que des méthodes, comme le transfert de valeurs, qui utilisent la connaissance de transfert sans l'évaluer peuvent avoir des résultats catastrophiques (transfert négatif). Cela a été montré par l'Annexe F (Figure 7.27). Le transfert de valeurs a des avantages quand on est sûr de la qualité de la connaissance de transfert, mais elle peut amener à pertes de performance si jamais la connaissance de transfert n'est pas très adaptée au problème.

Il faut également considérer l'inconvénient qui est de ne pas toujours pouvoir faire une copie des valeurs de la fonction d'utilité vers l'agent. Cette situation est même bien fréquente dans les problèmes du monde réel où la connaissance de transfert n'est pas décrite car elle est chez le spécialiste qui traite ce problème dans son quotidien.

Les méthodes que nous avons proposées (l'optimisation du transfert avec l'évaluation hors ligne et avec l'évaluation en ligne) ont eu une bonne performance pour toutes les expérimentations que nous avons réalisées. Nos méthodes ont eu les meilleurs résultats pour dix des dix-huit configurations. Et l'évaluation hors ligne a été meilleur sur quatre autres configurations cela fait en total de quatorze sur les dix-huit configurations expérimentées.

Les quatre autres configurations restantes ont été bien adaptées pour le transfert de valeurs qui a montré le meilleur résultat. Mais cela ne veut pas dire que nos méthodes ont eu un mauvais résultat. En fait, elles ne sont jamais loin derrière en attestant la bonne robustesse de nos algorithmes face aux différents types de connaissances de transfert.

En résumé, le transfert de connaissance peut apporter du bénéfice à l'apprentissage par renforcement. Les méthodes de transfert que nous avons proposés ont eu des bons résultats par rapport aux méthodes de transfert trouvées aujourd'hui dans la littérature. Nos méthodes ont prouvé sa robustesse en montrant une performance supérieure pour plus de la moitié des configurations que nous avons expérimentées et une performance très correcte pour les autres cas. Le grand avantage de nos méthodes est qu'elles ont ces bons résultats sans demander l'intervention de l'utilisateur. Nos méthodes s'adaptent à la qualité ou pertinence de la connaissance de transfert au problème traité, c'est-à-dire qu'ils utilisent beaucoup la connaissance de transfert quand elle est bien adaptée au problème ou ils l'ignorent quand la connaissance de transfert n'est pas adaptée.

Autrement dit, notre approche est robuste. Elle peut être moins bonne que les autres pour certains PDM, mais elle ne connaît pas de "zone de très mauvaise performance" contrairement aux autres.

4.12 Conclusions

Nous avons présenté dans ce chapitre notre contribution au transfert de connaissance en apprentissage par renforcement. Nous avons présenté, expérimenté et évalué notre algorithme en compagnie d'une comparaison entre les principales méthodes de transfert de connaissance en apprentissage par renforcement.

Etre générique et robuste sont les principaux avantages de notre algorithme, conforme montré dans ce chapitre notre algorithme ne demande aucune information de la part de l'utilisateur pour réaliser le transfert (en s'adaptent automatiquement à la qualité de la connaissance de transfert disponible) et quand il n'a pas été le meilleur il a obtenu un bon résultat dans tous les expérimentations réalisées en prouvant sa robustesse aux différents scénarios.

Les bons résultats avec le *gridworld* encouragent la continuation de la méthodologie choisi de commencer par un problème plus simple pour développer, évaluer et comparer notre algorithme de transfert pour ensuite l'appliquer à un problème plus complexe. Les prochains chapitres continuent donc cette approche d'aller vers un problème du monde réel et plus complexe.

Dans le chapitre suivant nous présentons le problème que nous allons traiter, l'état de l'art du domaine, notre motivation et objectifs. Dans le Chapitre 6, nous présentons comment nous avons conçu notre solution pour le problème en compagnie des expérimentations réalisées et les résultats obtenus.

Partie II :

**Application aux logiciels
d'entraînement sportif assisté par
ordinateur.**

Chapitre 5

Etat de l'art de l'entraînement sportif assisté par ordinateur

Ce chapitre présente la première utilisation de nos contributions au domaine du transfert de connaissance en apprentissage par renforcement dans un cas réel. Dans un premier temps nous allons survoler le domaine de l'entraînement sportif assisté par ordinateur. Dans un deuxième temps, nous détaillons la problématique de la simulation de situations de jeux. Cette partie est précédée de la motivation et définition du problème, en compagnie de sa problématique et l'approche utilisé.

Le reste du chapitre présente notre solution au problème de la simulation de situations de jeu. Cette solution est doublement originale : d'une part personne n'avait encore traité ce problème avec des techniques d'intelligence artificielle (systèmes multiagent, apprentissage automatique, etc.), et d'autre part il s'agit de la première application réelle des techniques d'optimisation du transfert de connaissance que nous avons créé.

Nous avons choisi le football comme première implémentation. Et nous avons donc développé une interface d'interaction avec l'entraîneur et un simulateur pour les expérimentations dans ce domaine. Néanmoins, ce travail est facilement applicable à d'autres sports d'équipe (par exemple, handball, hockey, volley, etc.).

5.1 Motivation et définition du problème avec l'approche envisagée

Grâce à une coopération scientifique avec le Laboratoire des Sciences de l'Information et des Systèmes (LSIS)¹, nous avons pu identifier le besoin d'outils et de méthodes d'informatiques sophistiqués pour la réalisation des recherches en STAPS (Sciences et Techniques des Activités Physiques et Sportives). Le besoin, le domaine d'application et l'interaction avec le groupe du LSIS nous ont motivé à développer un outil d'aide à l'entraîneur qui fait de la simulation de situations de jeu basé sur des techniques d'Intelligence Artificielle.

Cet outil a comme objectif de réaliser une simulation d'une situation de jeu décrite par l'entraîneur. Cependant l'objectif n'est pas que l'entraîneur décrive dans le moindre détail le déroulement de la situation de jeu. Nous envisageons plutôt un outil plus intelligent qui puisse inférer les informations manquantes à partir de la description donnée. De cette façon, l'entraîneur ne doit se concentrer que sur les points principaux, qui sont d'ailleurs les plus importants, sans avoir besoin de décrire ou donner tous les détails de l'évolution de la situation de jeu.

Ainsi, nous avons développé un simulateur avec des éléments plus autonomes qui vont prendre des centaines décisions pour exécuter les éléments de la situation de jeu qui ne sont pas décrits ou donnés. Dans cette simulation, les agents joueurs doivent exprimer un comportement qui soit cohérent avec la réalité, c'est-à-dire conforme aux restrictions et lois de la physique et imitent un comportement qui serait rationnel dans une situation réelle.

La problématique de la conception de cet outil repose principalement sur deux grands axes. Comment décrire une situation de jeu ? Et comment créer des agents suffisamment autonomes pour qu'ils puissent agir dans les situations qui ne sont pas prévues par la description ?

Le premier axe regroupe tous les problèmes d'interface avec l'utilisateur qui sont présents dans le domaine de la programmation par imitation (Bentivegna et Atkeson, 2002; Bentivegna *et al.*, 2002), programmation par démonstration (Atkeson et Schaal, 1997; Bakker et Kuniyoshi, 1996; Schaal, 1997). Quels sont les composantes de la situation de jeu qu'il faut préciser ? Et de quelle manière l'utilisateur peut-il décrire la situation de jeu ? Sont les questions pertinentes pour cet axe.

La deuxième problématique concerne l'autonomie des éléments présents dans la simulation. Quel niveau d'autonomie serait-il souhaitable ? Comment l'utilisateur pourrait

¹ <http://www.lsis.org/>

influencer cette autonomie ? Comment implémenter cette autonomie ? Comment ces éléments vont interagir entre eux ?

Pour répondre à toutes ces questions nous avons envisagé une approche basée d'une part sur une description graphique avec une ergonomie proche d'une description au tableau noir et des outils qui sont déjà utilisés par les entraîneurs, et d'autre part sur des agents autonomes dotés de la capacité d'apprentissage pour l'implémentation des éléments autonomes de la simulation. L'utilisation du transfert de connaissance pour l'apprentissage des agents a été motivée quand les chercheurs du STAPS nous ont décrit un système à base de règles élémentaires, qui fonctionne correctement pour une partie des situations de jeu.

Avant de présenter les simulations de situations de jeux dans le cadre d'aide à l'entraîneur, le problème que nous avons abordé et l'approche que nous avons choisie, nous présentons dans la section suivante la simulation sportive d'aujourd'hui et les principaux travaux liés.

5.2 Introduction aux systèmes d'aide à l'entraîneur

5.2.1 Le rôle de l'entraîneur et les schémas tactiques

Le déroulement d'un match de football (et d'autres sports) ne dépend pas uniquement des habiletés individuelles des joueurs, même si ce sont eux les responsables des actions sur le terrain. On ne peut pas négliger les choix stratégiques et tactiques adoptés par l'équipe, qui jouent un rôle très important pour le résultat final. Cette affirmation prend tout son sens quand une équipe considérée comme plus faible, du point de vue de la qualité individuelle des joueurs, gagne face à une équipe jugée plus forte.

La Coupe de l'Euro 2004 organisée par l'UEFA, où l'équipe de la Grèce a dépassé tous les favoris, a été un très bon exemple de l'importance du jeu tactique. Cette situation a mis en évidence le travail de l'entraîneur qui est responsable d'organiser tactiquement l'équipe, de définir la stratégie et de prévoir toutes situations de jeu et les réponses que l'équipe doit y apporter.

On appelle tactique l'ensemble des méthodes servant à gagner un conflit sur une petite échelle (au niveau local). Cela s'applique en particulier à la guerre (tactiques militaires) mais également à l'économie, au commerce, au jeu et à d'autres domaines comme la négociation. La tactique s'oppose à la stratégie qui est une action globale et à plus long terme.

Imaginons une circonstance où l'entraîneur et son équipe bénéficient d'un avantage au score ; il peut par exemple adopter la stratégie de bien placer ses joueurs, attendre l'attaque de

l'équipe adverse pour ensuite en profiter pour contre-attaquer. En même temps il peut choisir différentes options tactiques pour réaliser les contre-attaques : utiliser les passes courtes et rapides, se servir de passes longues pour les attaquants, ou encore utiliser les latérales du terrain, etc.

Nous venons de présenter quelques responsabilités inhérentes à l'entraîneur qui sont assez complexes. On peut maintenant imaginer la difficulté de se faire comprendre par tous les membres de son équipe, pour pouvoir ainsi les coordonner. En conséquence, l'entraîneur utilise des abstractions, dessins et schémas, pour pouvoir décrire ses idées et diminuer ainsi la difficulté de communication et les difficultés à se faire comprendre.

Il existe des logiciels conçus pour faciliter la communication de l'entraîneur, et celui-ci peut les utiliser pour se faire comprendre plus facilement. La suite de ce chapitre est consacrée à ce cadre.

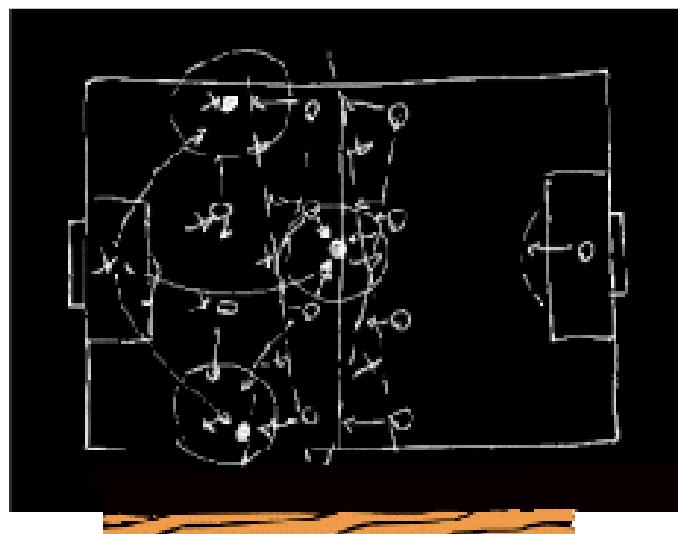


Figure 5.1 – Exemple d'une situation de jeu sur un tableau noir.

5.2.2 Les logiciels d'aide à l'entraîneur

Les entraîneurs utilisent communément des outils pour présenter leurs schémas tactiques. Les sportifs de tous niveaux connaissent bien le tableau noir (Figure 5.1) ou sa version magnétique. L'importance de ce type d'instrument est évidente pour présenter une séance d'entraînement ou pour préciser une tactique de jeu. Faire évoluer cet instrument est une démarche utile et nécessaire vu la complexité de la tâche de l'entraîneur à de se faire comprendre et l'état actuelle des outils utilisés aujourd'hui.

Utiliser l'ordinateur pour aider à réaliser une tâche quelconque est loin d'être innovant. On trouve d'ores et déjà un bon nombre de logiciels liés à cette pratique en éducation

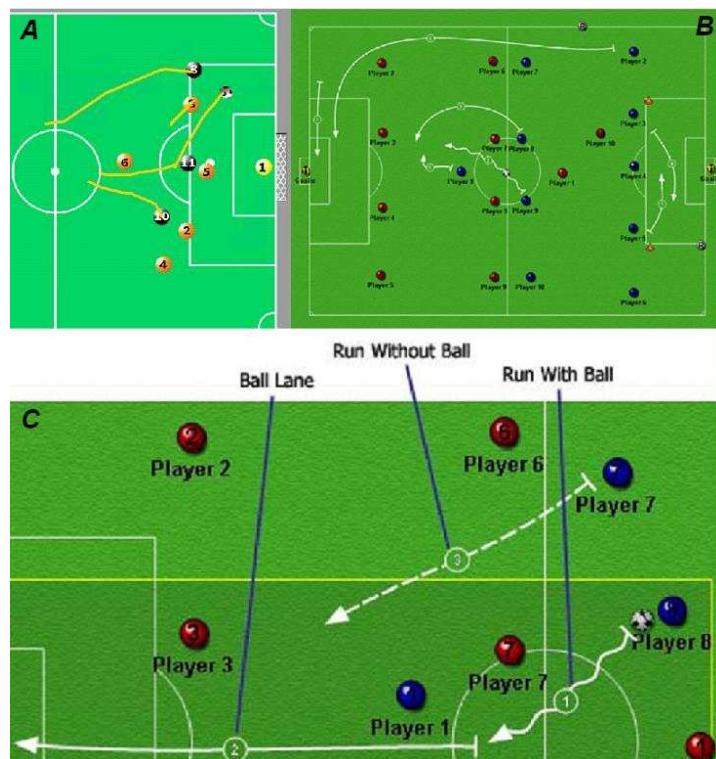
physique et sportive (EPS) qui proposent des solutions pour aider le travail de l'entraîneur. Au vu des logiciels existants aujourd'hui qui proposent une solution concernant les schémas tactiques, nous pouvons déterminer deux groupes distincts :

1. Ceux conçus pour remplacer le tableau noir ;
2. Et ceux qui font en plus de l'animation de schémas tactiques.

Le premier groupe n'apporte que peu de choses par rapport au tableau noir, ce sont plutôt des outils pour dessiner des schémas tactiques qui peuvent être enregistrés sur disque dur et utilisés ultérieurement. Les principaux avantages apportés sont donc :

- Avoir une assistance pour faire les dessins et une représentation plus attractive du point de vue esthétique ;
- Avoir la possibilité d'un répertoire de schémas tactiques, d'exercices d'entraînement ou de situations de jeu ;
- Un changement d'information plus facile grâce à la création de communautés virtuelles, groupes de discussion, bases de données, etc.

Pour donner une idée visuelle de ce que nous venons de présenter des copies d'écran de deux programmes du groupe 1, Campo Virtual et Data Coach, sont présentées sur la Figure 5.2.



Les logiciels qui font partie du deuxième groupe ont les mêmes caractéristiques que ceux du premier mais ils exploitent toutefois un peu plus l'ordinateur en ajoutant la possibilité d'animer le schéma. ForCoach, TactFOOT, Tatic Plus, Táticas et XSimul sont des représentants de ce groupe et quelques copies d'écran de ces programmes sont présentées sur la Figure 5.3.

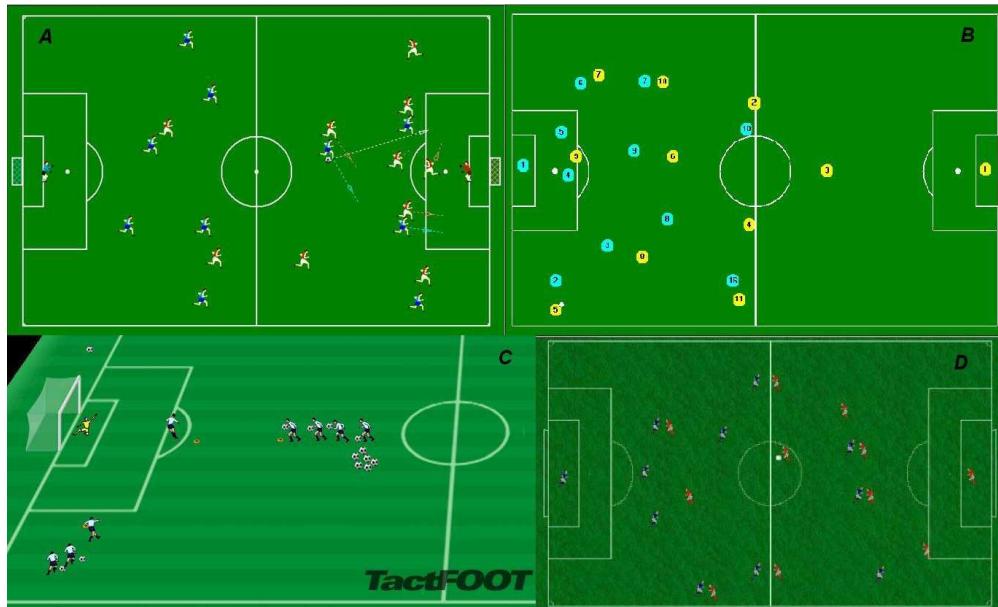


Figure 5.3 - Logiciels du groupe 2 :
(A)ForCoach ; (B)Tact Plus ; (C)TactFOOT ; (D)Táticas.

5.2.3 Limitation des logiciels d'aide à l'entraîneur

Malgré les avantages apportés, les logiciels mentionnés ont encore des limitations et des inconvénients qui sont importants pour le résultat final et pour l'utilisation. Le résultat final de l'animation est en général peu réaliste car des simples méthodes d'interpolation d'images sont utilisées. Pour avoir un résultat un peu plus proche de la réalité, il faut décrire quasiment tous les pas des joueurs/ballon et faire attention aux limitations du programme (quelques exemples de ces limitations sont montrés par la suite), ce qui complique considérablement le travail de l'entraîneur.

Pour donner un exemple de ce que nous venons d'évoquer dans le paragraphe précédent, imaginons que l'entraîneur veuille montrer un schéma ou un exercice où il faut passer entre deux balises et puis contourner un obstacle quelconque avec le ballon.

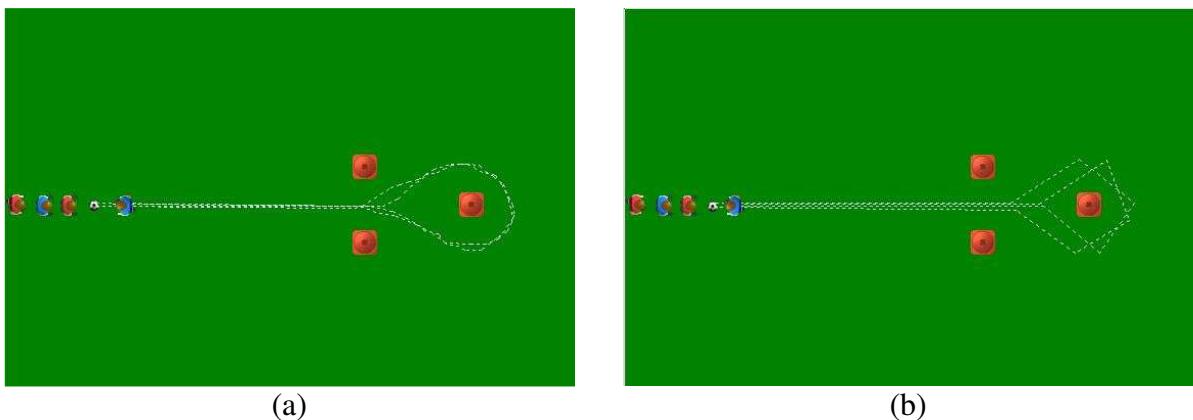


Figure 5.4 - Exemples de trajectoires :

(a) trajectoire plus proche de la réalité ; (b) trajectoire irréaliste.

Afin de réussir une animation convaincante, il faut préciser plusieurs points de la trajectoire du joueur et du ballon (Figure 5.4-a). Si jamais on ne précise qu'un nombre minimal de points, la trajectoire sera définie par des grands segments, ce qui n'est pas conforme à la réalité (Figure 5.4-b). Ces problèmes sont encore plus flagrants durant l'animation.

Les joueurs virtuels (agents) n'ont pas l'autonomie suffisante pour, par exemple, conduire le ballon par eux-mêmes, se déplacer à une vitesse inhérente à la situation, éviter les collisions, etc. L'entraîneur doit faire attention à tous les détails s'il veut un résultat fiable et réaliste. La Figure 5.5 et la Figure 5.6 illustrent deux de ces problèmes.

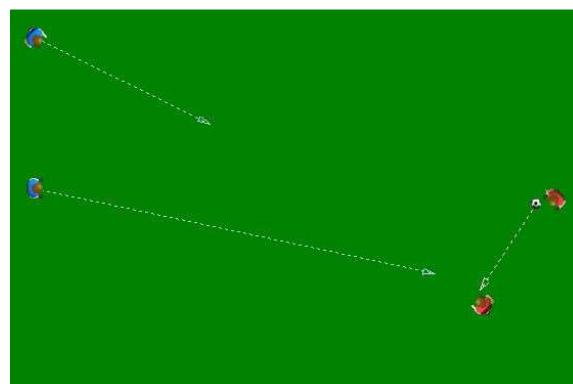


Figure 5.5 – Le problème de vitesse

Sur la première (Figure 5.5) on peut voir un problème de vitesse. Ce problème arrive quand l'utilisateur, des logiciels d'aujourd'hui, ne prend pas en compte l'ordre de grandeur de distances et du temps pour les parcourir. Dans la figure sont définis la trajectoire (par les flèches) de deux joueurs et du ballon entre deux situations définies par l'utilisateur. Cela veut dire que les joueurs et le ballon vont arriver au final de ses trajectoires au même temps.

Comme la distance que le joueur qui est au milieu et à gauche va parcourir est beaucoup plus grande que la distance attribuée aux autres, soit il sera très rapide par rapport aux autres, ou soit les autres seront très lents par rapport à lui, cas tous les éléments vont arriver au même moment à cause de la procédure d'interpolation.

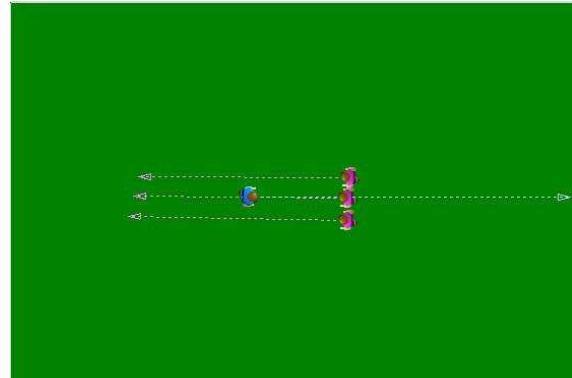


Figure 5.6 - Le problème d'obstacle et matériel

Une autre limitation de ces logiciels est qu'ils ne prennent pas en compte la présence/existence physique des éléments de la simulation. Ils peuvent par exemple passer les uns à travers les autres (comme montre la Figure 5.6) ou changer de direction sans avoir besoin de diminuer ou changer la vitesse.

Ces sont quelques limitations et inconvénients des logiciels existants et qui sont actuellement utilisés par les entraîneurs sportifs.

5.3 La simulation sportive d'aujourd'hui

Actuellement dès que l'on parle de simulation la majorité de gens pense tout de suite aux jeux vidéos, et par conséquent, le terme « simulation sportive » nous amène à ces types de jeux. Le terme simulation sportive est également utilisé par l'industrie du jeu vidéo pour définir le genre des jeux qui font allusion aux sports ou qui présentent cet univers ludique.

Dans le monde des jeux vidéo, il existe deux axes principaux : les jeux qui mettent le joueur à la place d'un sportif, et ceux qui placent le joueur dans un rôle de gestion ou de préparation de son équipe (par exemple : administrateur, gérant, sélectionneur ou entraîneur).

Cependant pour les applications dans un contexte sportif, il existe des travaux de recherche dans la sphère de l'Intelligence Artificielle et de la Robotique. Ces deux sous-domaines de l'informatique ont des qui simulent une confrontation sportive, opposant deux ou plusieurs camps, avec toutes les difficultés inhérentes à ce type d'environnement. Le but de

ces travaux est de concevoir, tester et développer des techniques d’Intelligence Artificiel et de Robotique.

Il existe plusieurs événements ou simulateurs qui ont comme objectif la création de robots sportifs, et dans divers sports. *Microsoft Robotics*¹ et *Webots*² ont des environnements pour le football, sumo et judo ; *EyeSim*³ et *TR Soccerbots*⁴ simulent le football et *RARS*⁵ simule une course d’automobiles. Ces sont de bons exemples d’événements ou de simulateurs, mais sans doute la *RoboCup*⁶ est, parmi tous les projets que nous venons d’énumérer le plus développé.

De façon synthétique, la simulation sportive d’aujourd’hui se situe dans deux domaines : les jeux vidéo et les simulateurs robotiques. Dans les sections suivantes, nous développons et présentons les principaux travaux de ces deux domaines qui sont liés à notre travail de recherche et au domaine choisi.

5.3.1 Les jeux vidéo de simulation sportive

L’objectif de cette section n’est pas d’être exhaustif sur le sujet des jeux de simulation sportive mais de montrer son état actuel et de présenter en peu de mots les techniques utilisées dans le monde des jeux vidéo. Au vu de cet objectif nous allons présenter la partie la plus pertinente dans le cadre de ce travail de recherche, c’est-à-dire les techniques d’Intelligence Artificielle (IA) les plus utilisées par l’industrie des jeux vidéo. Nous présentons ensuite comme ces techniques sont utilisées avec les jeux de simulation sportive.

La première chose qu’il ne faut pas oublier quand on parle de jeux vidéo est qu’ils sont des produits commerciaux qui n’ont pas pour objectif ultime gagner sur l’utilisateur mais de le divertir. Les développeurs suivent complètement ce principe pour la conception des jeux vidéo.

Pour simplifier, nous pouvons diviser un jeu vidéo en trois parties : la simulation, l’IA et les entrées de l’utilisateur (sachant que ces dernières ne sont pas obligatoires). La simulation fournit l’environnement et les règles de cet environnement. L’IA est responsable du comportement ou des actions des agents qui ne sont pas contrôlés par l’utilisateur. Et les entrées d’utilisateurs sont les commandes données par l’utilisateur.

¹ Microsoft Robotics – <http://msdn.microsoft.com/en-us/robotics/default.aspx>

² Webots – <http://www.cyberbotics.com>

³ EyeSim – <http://robotics.ee.uwa.edu.au/eyebot/doc/sim/sim.html>

⁴ TR Soccerbots – <http://www.trsoccerbots.org>

⁵ RARS – Robot Auto Racing Simulator – <http://rars.sourceforge.net>

⁶ RoboCup – <http://www.robocup.org>

La simulation (animation graphique) est sans doute la partie du jeu vidéo qui reçoit la grande majorité des investissements et du temps de développement : les résultats en sont marquants. Avec l'avancement de la partie graphique l'IA est devenue de plus en plus importante. Les joueurs n'aiment pas du tout quand les agents du jeu ont un comportement absurde. Il s'agit d'un jeu mais les joueurs veulent avoir l'impression qu'il existe quelque chose d'intelligent de l'autre côté. L'IA peut faire des erreurs mais il ne faut pas que ces erreurs soient stupides. En fait, les joueurs aiment quand l'IA joue bien et qu'elle n'est pas parfaite, ce qui renforce encore plus le principe de divertissement. (Dybsand *et al.*, 2001; Woodcock, 2000)

Pour implémenter cette IA les développeurs de jeux vidéo utilisent communément des scripts, machines à états, systèmes experts, de la logique floue ou des techniques d'apprentissage.

La *programmation avec des scripts* est conceptuellement simple, en fait elle est très proche du travail d'un dramaturge¹, qui décrit les actions de ces personnages. De toutes les méthodes utilisées par les développeurs de jeux vidéo, les scripts peuvent donner des résultats qui peuvent aller du plus intelligent au plus inepte. Lorsque les situations sont proches de ce que l'auteur du script avait prévu, l'IA peut apporter des résultats très intelligents. Lorsque le monde simulé est loin de ce que l'auteur avait envisagé, les résultats peuvent être catastrophiques (Dybsand *et al.*, 2001).

Les *machines à états* organisent le comportement de l'IA d'une manière plus formelle qu'un script. Les développeurs associent un comportement à un état de la machine à états, et déterminent les transitions entre les états. L'IA va donc évaluer ce qu'il faut faire en fonction de l'état actuel et aussi évaluer s'il faut changer d'état. Par exemple, dans un jeu de guerre une unité peut avoir les états : fuite, marche, combat et poursuite. L'IA va donc implémenter le comportement pour chaque état et les transitions entre eux. L'inconvénient de cette approche est la croissance du nombre d'états qui augmente avec le nombre d'entrées, si jamais le nombre d'entrées augmente le nombre d'états peut exploser. Cependant les machines à états sont faciles à implémenter et donnent souvent des résultats intéressants pour les jeux vidéo(Dybsand *et al.*, 2001).

Les *systèmes experts* essayent de reproduire la connaissance d'un expert. Le système a un ensemble de règles du type « *si condition, alors faire quelque chose* ». Les conditions des règles sont comparées avec l'état du monde et celles qui sont vraies vont contribuer au

¹ Personne qui écrit pour le théâtre

comportement de l'agent. L'utilisation de règles est une option pour résoudre le problème de l'explosion du nombre d'états d'une machine à état, parce qu'une condition peut représenter plusieurs états. L'inconvénient de l'utilisation des systèmes experts est le *debuging* des systèmes avec un gros nombre de règles (Dybsand *et al.*, 2001).

Les systèmes experts, et d'autres systèmes basés sur la logique booléenne, peuvent être étendus avec l'introduction de la *logique floue*. Celle-ci utilise les conditions de la logique booléenne classique « si condition vraie alors » et ajoute des intervalles et, par exemple, le concept de « assez proche ». Les règles deviennent donc clauses du type « si condition assez proche alors » (Dybsand *et al.*, 2001).

Il existe une variété de *techniques d'apprentissage* qui sont parfois mentionnées dans les conférences de développeurs de jeux vidéo qui ont comme centre d'intérêt l'IA. Cependant l'apprentissage automatique est rarement présent dans les jeux vidéo. *Black & White*¹, *Colin McRae Rally 2.0*², *Forza Motorsport*³, *Quake*⁴, *Creatures*⁵ et *Conflict Zone*⁶ sont des rares exemples d'utilisation de l'apprentissage. Ces jeux ont employé des techniques d'apprentissage/évolution comme : les réseaux de neurones, les arbres de décision, les algorithmes génétiques, la programmation génétique et les modèles probabilistes.

Ce que nous venons de présenter sont des exemples de techniques utilisées par l'industrie de jeux vidéo pour implémenter l'IA. Cela donne une idée de l'état actuel de l'IA dans les jeux, mais il existe un grand nombre de développeurs qui « trichent » au moment de la création de l'IA des jeux. La triche (*cheating*) est le terme du jargon des jeux vidéo pour décrire la situation où l'IA a des priviléges par rapport au joueur. Comme par exemple : plus de points de vie, une vitesse supérieure, ou la méconnaissance du brouillard de guerre (voile qui obscurcit une partie de la carte et empêche le joueur de connaître les activités de l'ennemi).

La possibilité de tricher vient du fait que le but ultime des jeux est le divertissement et ils n'ont pas du tout l'obligation de respecter le modèle de l'environnement. Elle est couramment utilisée pour créer un défi plus grand à l'utilisateur ou pour simplifier l'implémentation de l'IA. L'opinion commune des développeurs par rapport à la triche est : tout d'abord essayer de construire une bonne IA avant de recourir à la triche ; ensuite, être sûr que l'utilisateur sache que la triche sera présente avec l'augmentation du niveau de difficulté ;

¹ *Black & White* – Lionhead/Electroinc Arts – <http://www.lionhead.com/bw2>

² *Colin McRae Rally 2.0* – Codemasters – <http://www.codemasters.com/cmr2pc>

³ *Forza Motorsport* – Microsoft Games – <http://forzamotorsport.net>

⁴ *Quake* – id Software/Activision – <http://www.idsoftware.com/games/quake>

⁵ *Creatures* – Creature Labs/Mindscape – http://www.gamewareddevelopment.co.uk/creatures_index.php

⁶ *Conflict Zone* – Masa/Ubisoft – <http://www.conflictzone-thegame.com/>

et enfin, essayer au maximum de ne jamais laisser l'utilisateur voir l'IA tricher (Dybsand *et al.*, 2001).

Pour parler plus précisément des techniques utilisées par les jeux de simulation sportive, nous voudrions référencer les plus grands titres de l'actualité qui s'approchent le plus du type de situation que nous allons étudier dans le prochain chapitre. Les jeux intitulés *Madden NFL*¹, *NBA Live*², *NHL*³, *FIFA Soccer*⁴ et *Pro Evolution Soccer*⁵ sont les représentes le plus importants. Mais cette proximité se résume à avoir un groupe d'agents qui participent à une partie d'un match. L'objectif et les techniques qui implémentent le raisonnement ne sont pas du tout les mêmes.

Le code source des jeux commerciaux étant confidentiel il n'est pas possible de connaitre dans le détail les techniques d'IA appliquées. En revanche, il existe des indices, des déclarations et quelquefois des publications qui mentionnent ce qui est utilisé par les développeurs de jeux vidéo.

D'une manière générale, l'IA des jeux de sport est structurée en couches de la même façon que pour les jeux du type « *First-Person Shooter* » (FPS⁶) ou les jeux de stratégie en temps réel (RTS⁷). Les couches le plus basses sont responsables de l'animation et du déplacement ; au milieu nous avons une couche pour la prise de décision individuelle et au niveau le plus haut le raisonnement tactique. Toutefois, il y a de grandes différences dans chacune de ces couches quand on s'adresse à d'autres types de jeu (Champandard, 2008).

L'animation/déplacement est la couche qui fournit les résultats le plus attendus par les joueurs. Cette partie est considérée de loin comme la plus importante pour ce type de jeu, qui exige un haut niveau de réalisme pour le mouvement des agents du jeu, pour la manipulation de la balle, ou même l'interaction avec les autres agents. Tout le reste est construit et basé sur cela.

L'implémentation de la couche de prise de décision individuelle utilise plusieurs comportements de navigation (*steering behaviors*) (Reynolds, 1999) ; elle conduit l'agent en combinant des comportements de base. Nous avons utilisé cette technique pour implémenter le déplacement des nos agents sur le terrain. Pour plus de détail voir l'Annexe B. Le reste de

¹ *Madden NFL* – EA Sports/Eletroinc Arts – <http://www.easports.com/madden08>

² *NBA Live* – EA Sports/Eletroinc Arts – <http://www.easports.com/nbalive08>

³ *NHL* – EA Sports/Eletroinc Arts – <http://www.easports.com/nhl09>

⁴ *FIFA Soccer* – EA Sports/Eletroinc Arts – <http://www.fifa08.com>

⁵ *Pro Evolution Soccer* – Konami – <http://fr.games.konami-europe.com>

⁶ De l'anglais : *First-Person Shooter* : tir à la première personne

⁷ De l'anglais : *Real-Time Strategy*

l'IA de cette couche utilise une technique simple de prise de décision, comme les machines à états finis (Champandard, 2008), pour exécuter la tactique issue de la couche au-dessus.

La définition du comportement des agents est un problème qui n'est pas du tout évident à résoudre si jamais on garde une approche puriste en respectant les règles du modèle. Cependant la majorité des jeux vidéo utilisent une part de triche pour implémenter cette couche. Par exemple, ils collectent des informations de l'entrée de l'utilisateur et en profitent pour choisir parmi les tactiques prédéfinies (par exemple : utiliser l'information que l'utilisation a fait la commande de tourner à gauche pour implémenter les actions des adversaires). Tout est beaucoup plus simple si jamais on connaît les actions de l'utilisateur avant qu'elles soient exécutées dans l'environnement que essayer de modéliser le comportement de l'adversaire ou de le prévoir.

5.3.2 La simulation d'équipes sportives, l'exemple de la *RoboCup*

En dehors de l'univers des jeux vidéo, il existe des travaux qui font de la simulation informatique ou de la simulation numérique, ces derniers font des expérimentations en respectant un modèle d'environnement défini. Dans ce cas le divertissement ne fait pas partie du projet, la « triche » n'est pas du tout acceptée et les efforts de création de l'interface avec l'utilisateur sont plutôt dirigés vers la présentation des résultats. Afin de laisser les résultats compréhensibles, parfois cela peut être un simple tableau ou une courbe.

Un bon exemple de ce type de simulation qui utilise un environnement sportif pour les expérimentations est la *RoboCup* (Kitano *et al.*, 1995). La *RoboCup*, qui était appelée à l'origine de *Robot World Cup Initiative*, est un projet de coopération international destiné à encourager le développement de l'Intelligence Artificielle (IA), de la robotique et d'autres domaines connexes. Cette initiative est une tentative de stimuler la recherche dans ces domaines en fournissant un problème standard où une large gamme de technologies peut être mise en œuvre et étudiée.

Dans ce cadre-là, la *RoboCup* a choisi le football comme principal domaine d'application. Une coupe du monde de football de robots (*The Robot World Cup Soccer Games*) est organisée de même que la conférence où sont présentés et publiés les travaux et les résultats obtenus. Au moment de sa création, le grand rêve ou le dernier objectif que les inventeurs de la *RoboCup* avaient, était que vers la moitié du XXI^e siècle une équipe de robots footballeurs humanoïdes complètement autonomes puisse gagner, en obéissant aux règles officielles de la FIFA, face au vainqueur de la dernière Coupe du Monde.

Actuellement la *RoboCup* a trois grands domaines de compétitions: des compétitions de football (*RoboCupSoccer*), des compétitions de sauvetage (*RoboCupRescue*) en utilisant robots réels et simulations et des compétitions consacrées aux jeunes étudiants (*RoboCupJunior*)

Les différentes catégories de compétitions de football, affiliées à la *RoboCup*, existantes ou proposées sont:

- la catégorie des petits robots (moins de 180 centimètres carrés), en comprenant jusqu'à cinq robots par équipe ;
- la catégorie des robots de taille moyenne (d'environ 250 centimètres carrés), en comportant jusqu'à onze robots par équipe ;
- la catégorie des robots Sony quadrupèdes, comportant trois robots par équipe ;
- la catégorie des robots humanoïdes, en cours de définition ;
- la catégorie simulation, en comprenant onze programmes par équipe plus un agent entraîneur en option.

Soccer server est le nom donné au simulateur qui permet de faire jouer, au football, différentes équipes d'agents. Il a été développé comme une plate-forme de tests pour les systèmes multiagent qui évoluent dans un environnement dynamique et conforme au monde réel. La plate-forme fournit un terrain de jeu virtuel et simule les mouvements des joueurs et de la balle.

Chaque année, le nombre d'équipes de joueurs artificiels connaît une progression constante, et la plupart d'entre elles font appel à un grand nombre de techniques d'IA différentes. Cependant ces évolutions ne sont pas très orientées vers les professionnels du sport et les résultats ne sont pas encore appliqués à l'entraînement sportif. Cela viendra peut-être avec l'avancement des recherches, mais pour l'instant les équipes de robots (réels ou simulés) sont plutôt concentrés sur l'efficacité face à l'adversaire et la victoire du match.

5.3.3 La simulation sportive d'aujourd'hui et l'entraînement sportif

D'après ce que nous venons d'exposer dans les sections précédentes, nous pouvons voir que : les travaux qui sont directement liés à l'entraînement sportif ne sont pas encore suffisamment évolués pour remplir tous les besoins des professionnels du sport ; et les travaux qui font de la simulation sportive ne sont pas adressés au domaine de l'entraînement sportif.

Si d'un côté les jeux vidéo se sont beaucoup intéressés à l'animation et immersion de l'utilisateur dans le jeu, afin d'augmenter le divertissement, de l'autre ils n'ont pas trop

développé le comportement des agents d'une façon puriste (sans tricher). Dans ce domaine, le but ultime est le divertissement des utilisateurs, peut importe comment les agents sont développées.

Les travaux en simulation se sont, pour l'instant, intéressé à créer, développer et tester des nouvelles techniques dans le domaine de l'IA et de la Robotique. Ils ne se sont pas encore approchés du milieu de l'entraînement sportif, en essayant d'appliquer ces techniques dans le quotidien des professionnels du sport.

Notre objectif n'est pas du tout de gagner un match face à un adversaire (comme la *RoboCup*) et non plus de divertir l'entraîneur avec un haut niveau de graphisme/animation et un déplacement créé sans respecter les règles de l'environnement simulé (comme les jeux vidéo). Mais de réaliser sur le terrain ce qui a été indiqué par l'entraîneur. Le travail que nous avons développé sera présenté par la suite.

Chapitre 6

Conception d'une équipe d'agents apprenants pour la simulation de situations de jeu

Ce chapitre présente notre approche pour la problématique de simulation de situations de jeu. Le problème de la description d'une situation de jeu est traité par une représentation abstraite et visuel. De cette façon l'utilisateur peut faire sa description d'une manière simple et sans donner tous le détail de la situation de jeu.

Cependant pour pourvoir utiliser cette abstraction comme description de la situation de jeu, nous avons besoin de joueurs assez autonomes pour agir dans les situations ou parties qui n'ont pas été ajoutées à la description. Et nous avons également besoin de joueurs qui soient capables de s'adapter au comportement de l'adversaire ou au changement de l'environnement.

Dans un premier temps, nous allons présenter notre approche pour le déploiement de situations de jeux. Ensuite, nous présentons la manière dont nous avons traité l'interaction

avec l'utilisateur et comment il va décrire la situation de jeu. Puis, nous allons décrire la conception de nos agents.

6.1 Une approche pour le déploiement autonome et adaptable de situations de jeux

Une fois que nous avons présenté la simulation sportive d'aujourd'hui et les principaux travaux liés. Nous pouvons reprendre l'approche que nous avons envisagée pour le problème de la simulation de situations de jeu, et détailler les idées que nous avons commencé à parler dans la section 5.1.

D'un point de vue bien pragmatique, ce que nous proposons est un outil qui soit plus intelligente que les outils qui sont actuellement utilisées par les professionnels du sport (cf. section 5.2.2). C'est-à-dire, un outil plus autonome qui puisse inférer les informations manquantes à partir de la description donnée, sans demander que cette description soit trop détailler. De cette manière, l'entraîneur ne va fournir que les points importants de la situation de jeu.

D'une manière plus scientifique nous devrons répondre aux différents points de la problématique se la section 5.1. Ainsi pour faire face au premier grand axe de la problématique : comment faire la description de la situation de jeu. Nous nous sommes inspirés du tableau noir/tablette utilisé par le coach et des techniques utilisées dans les logiciels de création d'applications multimédia. Nous avons donc créé une interface qui permet la description d'une situation de jeu en utilisant la souris. Cette description consiste à placer les éléments de la situation de jeu (joueurs et objets) sur leur position de départ et ensuite créer des frames (instantanés) du déroulement souhaité par l'entraîneur jusqu'au dernier frame, autrement dit comme la trame de la situation de jeu.

Le premier frame précise la situation de départ de la simulation (positionnement de tous les éléments). Les autres frames sont plus souples et l'entraîneur peut définir des zones du terrain par où les joueurs doivent se placer portant le ballon ou pas. Cela veut dire donc que nous pouvons définir une situation de jeu comme une séquence avec un minimum de deux frames. Où le premier frame définit la situation de départ de la simulation et les autres frames définissent l'ordre des régions du terrain par où les agents doivent passer.

Avec la description de la situation de jeu définie, nous avons affronté le deuxième grand axe de la problématique : comment créer des agents assez puissants pour réaliser une situation de jeu décrite selon la description choisie. Etant donné le problème que nous affrontons, il

nous a apparu évident le besoin d'avoir agents assez autonomes pour que les mêmes puissent savoir ce qu'il faut faire entre deux frames de la description de la situation de jeux.

Pour construire ces agents avec le niveau d'autonomie souhaité et réaliser la simulation avec la description des situations de jeu définie, nous avons utilisé les concepts et techniques des Systèmes Multiagent (SMA) qui nous semblent bien adaptées pour le problème.

Afin d'éviter la difficile tâche d'implémenter le comportement des agents avec toutes les réactions qui sont inhérentes à un joueur, nous avons ajouté de l'apprentissage automatique à nos agent. De cette manière, nous n'avons pas besoin de prévoir et implémenter un comportement pour toutes les situations possibles. La technique d'apprentissage automatique que nous avons utilisé a été l'apprentissage par renforcement. De cette manière les agents vont apprendre par essais et erreurs comment agir face à les situations de jeu et ils sauront s'adapter aux changements ou à des nouvelles situations de jeu.

Nous considérons l'autonomie et l'adaptabilité deux autres avantages importants apportés par les SMA et par l'apprentissage automatique. Les agents sont plus autonomes parce qu'ils apprennent les détails que l'entraîneur n'a pas donnés. Et ils sont capables de s'adapter aux changements de la situation de jeu et des adversaires.

Cependant, cette approche a l'inconvénient d'avoir un temps d'apprentissage pour commencer à présenter des bons résultats. Dans l'intention de réduire ce temps d'apprentissage et d'avoir des bons résultats le plus vite, nous avons appliqué les techniques d'optimisation du transfert de connaissance que nous avons créé (cf. Chapitre 4).

6.2 Interaction avec l'utilisateur

Une des premières choses qu'il faut pour pouvoir simuler une situation de jeu est de connaître la situation de jeu à simuler. Pour cela il faut que l'utilisateur du système puisse la donner. Et lorsque celle-ci n'est pas sous le format accepté par le système il faut qu'il puisse la créer, ou la décrire.

Dans cette optique nous avons opté par une approche visuelle. C'est-à-dire que l'entraîneur réalise toute sa description en utilisant la souris pour placer sur le terrain les éléments qui font partie de la situation de jeu. Puis il détermine l'enchaînement des séquences de jeu appelées frames qui vont décrire les zones du terrain par lesquelles les joueurs doivent passer et ce avec ou sans le ballon. La Figure 6.1 montre la description d'un une-deux en utilisant trois frames. Dans le frame *a*, l'utilisateur place les joueurs sur leurs positions de

départ pour ensuite définir les deux autres frames avec les zones (représentées par des carrés) par lesquelles les agents doivent passer.

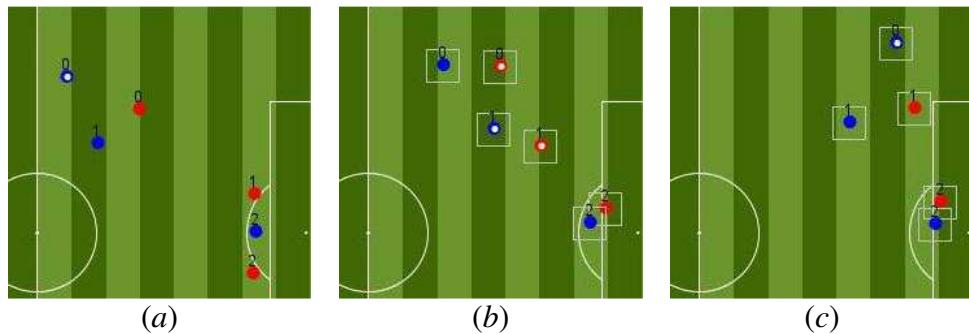


Figure 6.1 – Exemple d'une description d'un une-deux

Nous pouvons remarquer que le joueur 0 de l'équipe bleu commence avec le ballon, ensuite le joueur 1 doit atteindre sa zone avec le ballon (Figure 6.1(b)) et enfin le joueur 0 doit récupérer le ballon pour atteindre son deuxième objectif. Les zones avec le ballon attribuées aux joueurs 0 et 1 de l'équipe rouge vont leur obliger à aller chercher le ballon, étant donné qu'ils ont besoin du ballon pour accomplir ses objectifs.

Ce que nous venons de montrer illustre bien la différence entre notre approche et la fastidieuse action de décrire de tous les frames de la situation de jeu utilisé par les outils actuelles. Nous arrivons à décrire une situation comme un une-deux avec trois frames.

D'une manière plus général la description consiste à attribuer à chaque joueur i , une séquence de couples $\langle(z_1, b_1), (z_2, b_2), \dots, (z_n, b_n)\rangle$ où z_i représente une zone sur le terrain (par exemple un carré à une position donné et de taille donnée) et b_i est un booléen qui indique si le joueur doit passer dans la zone avec le ballon ou pas.

Avec la description de la situation de jeu définie nous pouvons passer à la conception des agents.

6.3 Conception des agents

Une fois que nous avons la description de ce que l'entraîneur veut simuler, il nous manque des « joueurs » (agents) capables de le réaliser. Comme notre approche est d'alléger la description donnée par l'entraîneur de manière à ce qu'elle ne soit pas fastidieuse à renseigner, il faut que nos agents soient capables de déployer les situations de jeu à partir de ce niveau de détail.

Avec l'approche de la section précédente, nous avons simplifié la description d'une situation de jeu par rapport aux outils utilisés aujourd'hui par les professionnels du sport. La section suivante présente la conception et le développement des agents « joueurs » qui suivront la description de la situation de jeu considérée.

6.3.1 Déplacement et actions

Avant de commencer à parler du raisonnement des agents et de leur comportement face aux coéquipiers et adversaires, il faut traiter la partie physique/motrice et déterminer quelles sont les actions possibles pour les agents.

En ce que concerne la partie physique/motrice nous nous sommes inspirés des solutions utilisées par les animations, par le cinéma et par les jeux vidéo. D'une manière plus précise nous avons utilisé le modèle *boids* avec ses comportements de navigation (*steering behaviors*) (Reynolds, 1999). Ce modèle a été originalement créé pour simuler les comportements trouvés dans les vols d'oiseaux ou les bancs de poissons. Il est fondé sur des comportements élémentaires : s'aligner, se rassembler, se séparer, etc. Et plusieurs autres comportements peuvent être créés à partir de la combinaison des comportements élémentaires. Ces comportements et le modèle sont détaillés dans l'Annexe B.

Les *steering behaviors* possèdent encore un *modèle de vision*, un *modèle de collision* et un *modèle de déplacement* basés sur les lois de la Mécanique, ce qui rapproche le déplacement des agents de celui qui peut être réalisé dans le monde réel. C'est-à-dire que les vitesses, accélérations et changements de direction vont respecter les lois de la Physique. Les comportements de navigation ont été déjà utilisés par (Helbing *et al.*, 2000; Hess et Modjtahedzadeh, 1989; Jun *et al.*, 2006; Yu, 2007) dans la construction de ses agents pour bien représenter le déplacement réalisé par les humains.

Le *modèle de vision* est simple et définit un champ de vision limité selon ce que l'on veut simuler. Le *modèle de déplacement* détermine le centre de gravité, masse, position, vitesse, orientation, force maximale et vitesse maximale de l'agent ainsi que tous les mécanismes pour calculer les changements de position, vitesse, orientation etc. Et le *modèle de collision* permet de détecter les éventuelles collisions entre les agents.

6.3.1.1 L'espace d'actions

Avec le modèle de déplacement définit et en discutant avec des professionnels du sport nous avons pu définir les actions élémentaires des joueurs. Par conséquence, nous avons les utilisées pour la conception des nos agents. Ces actions sont les suivantes :

- *Se déplacer vers un point* : nous avons réalisé cette action-là en combinant les comportements *seek* et *unaligned collision avoidance* afin d'aller vers un point en évitant les collisions ;
- *Se déplacer en groupe vers un point* : cette fois nous avons ajouté les comportements *cohesion* aux deux autres présents dans l'action précédente, cette action permet de rester proche du groupe considéré ;
- *Se positionner* : inspiré de l'action se positionner de (Veloso *et al.*, 1999), nous combinons les comportements de *cohesion*, *separation* et *unaligned collision avoidance*. Cette action va déplacer l'agent vers une zone stratégique qui est en même temps proche de ses compagnons et loin de ses adversaires, toujours en évitant les collisions ;
- *Marquer un adversaire* : le joueur utilise une heuristique pour déterminer un point proche d'un adversaire qui soit entre le but et le ballon. Ensuite il se dirige vers ce point comme pour l'action se déplacer vers un point ;
- *Intercepter le ballon* : en utilisant l'algorithme de calcul numérique (Stone et McAllester, 2001) pour trouver le temps pour intercepter le ballon, avec cette information l'agent peut savoir vers où il doit aller pour attraper le ballon ;
- *Faire une passe* : tir le ballon vers un agent en considérant sa position et sa vitesse ;
- *Prendre le contrôle du ballon* : essai de contrôler le ballon, mais le succès dépend des conditions de l'environnement.

Dans la section suivante nous présentons comment ces actions sont utilisées par nos agents.

6.4 Raisonnement et la sélection d'actions

Lors d'une réunion avec M. Mohamed SEBBANE, qui est entraîneur et prépare une thèse de doctorat au LSIS à Marseille, nous avons appris qu'il y a des règles et principes généraux qui sont bien définis, clairs et facile à comprendre. En revanche, l'exécution sur le terrain n'est pas du tout évidente.

Ces principes disent qu'il y a deux situations distinctes pour une équipe : soit l'équipe est en attaque, soit elle est en défense. Une fois qu'il a eu une perte du ballon, il y a trois principes généraux : se remplir défensivement ; se placer pour défendre ; remonter sur le joueur qui a le ballon. Une fois que l'équipe a récupéré le ballon : arriver le plus rapidement

possible au but de l'adversaire ; faire une attaque placée ; se positionner si on n'a pas le ballon ;

Lié à des principes plus généraux comme ceux que nous venons de montrer, il y a des petites règles comme : ne jamais faire une passe latérale dans la zone défensive, le défenseur doit se positionner dans un triangle avec deux attaquants parmi lesquels un a le ballon, etc.

Après la réunion nous avons été convaincus qu'il ne serait pas du tout facile d'implémenter un script pour suivre les principes présentés par l'entraîneur. Cela vient du fait que l'exécution de ces principes change énormément selon la situation du jeu. Cette difficulté est prise en compte pour le choix de l'utilisation de l'apprentissage automatique pour constituer le comportement des agents. Il est bien difficile de prévoir toutes les situations qu'un joueur peut trouver et en plus déterminer la meilleure action à prendre dans tous les cas. Et l'utilisation de l'apprentissage par renforcement va nous aider pour cette tâche. La section suivante montre comment nous avons réalisé l'apprentissage par renforcement pour la tâche de la simulation d'une situation de jeu donnée.

6.4.1 Apprendre par renforcement

Face aux difficultés de conception d'un programme qui puisse représenter le comportement des agents pour toutes les situations possibles, nous avons opté pour l'utilisation des méthodes d'apprentissage automatique pour acquérir ces comportements d'une façon automatique, sans avoir besoin donc de les développer directement. Cela va ainsi supprimer la complexe tâche de programmer les comportements des joueurs, et va également ajouter de l'adaptabilité aux agents.

La technique d'apprentissage automatique que nous avons utilisée a été l'apprentissage par renforcement. Elle nous est apparue la plus adaptée au problème de la simulation d'une situation de jeu et nous a permis d'utiliser et valider notre algorithme d'optimisation du transfert de connaissance dans un cas réel.

La suite de cette section montre comment nous avons réalisé l'apprentissage par renforcement et le transfert de connaissance. C'est-à-dire : comment nous avons représenté les états ; comment nous avons défini les récompenses et construit l'apprentissage ; et comment nous avons réalisé le transfert. Les expérimentations réalisées et les résultats obtenus seront montrés dans la section 6.6.

6.4.1.1 Représentation de l'espace d'états

Pour représenter les états nous avons profité de la symétrie du terrain en faisant une représentation qui n'est pas liée à des points cartésiens (x, y) mais qui utilise des distances et angles. Cela permet de réutiliser ce que l'on a appris sur une partie du terrain sur une autre partie. Un agent va par exemple pouvoir tirer un avantage de ce qu'il a appris sur le côté gauche du terrain quand il sera du côté droit.

Ce type de représentation est une pratique courante dans la communauté de l'apprentissage par renforcement (Andou, 1998; Stone et Sutton, 2001). Pour représenter un état nous avons pris un vecteur de 12 éléments qui contient des distances et angles entre des choses importantes pour la situation de jeu comme : l'objectif, les buts, le ballon, l'adversaire le plus proche, le coéquipier le plus proche ; et des informations essentielles comme quelle équipe a le ballon etc.

Le Tableau 6.1 détaille chaque élément du vecteur que nous avons utilisé pour représenter un état.

Caractéristiques de l'état	
1	distance entre l'objectif de l'agent et le but de l'adversaire
2	distance entre l'objectif de l'agent et son but
3	distance entre l'agent et son objectif (-1 s'il n'a pas d'objectif)
4	distance entre l'agent et le but de l'adversaire
5	distance entre l'agent et le son but
6	distance entre l'agent et l'adversaire le plus proche (-1 s'il n'a pas d'adversaire)
7	distance entre l'agent et le ballon
8	si l'agent a le ballon ou pas
9	quelle équipe a le ballon (-1=adversaire ; 0=personne ; 1=son équipe)
10	si l'agent est sensé arriver à l'objectif avec le ballon ou pas
11	distance entre l'agent et le compagnon T_i (-1 s'il n'a pas d'adversaire)
12	angle ¹ entre l'axe x et le compagnon T_i ($[0 ; 2\pi]$ ou -1 si T_i n'existe pas)

Tableau 6.1 – Vecteur qui représente un état.

La Figure 6.2 montre graphiquement quelques éléments de notre vecteur. Les numéros indiqués montrent la position de l'élément dans le vecteur du Tableau 6.1, par exemple le numéro 1 est la distance entre l'objectif de l'agent et le but de l'adversaire, le numéro 2 la distance entre l'objectif de l'agent et son but, etc.

¹ Dans le sens inverse des aiguilles d'une montre.

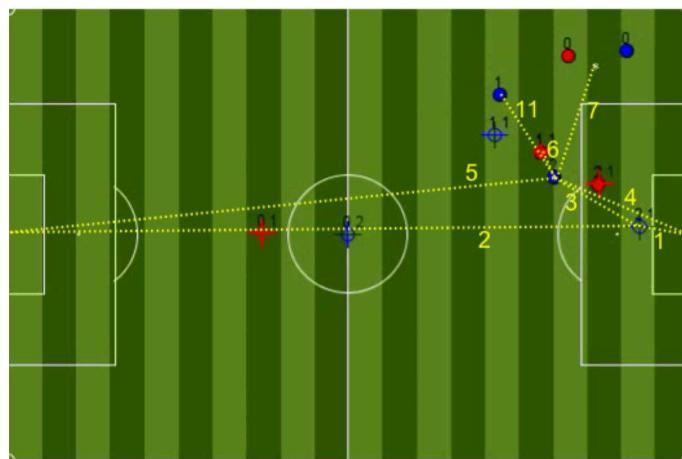


Figure 6.2 – Présentation graphique des distances.

Les valeurs des éléments du vecteur sont adaptées à chaque agent. Autrement dit, chaque agent a sa propre vision du monde.

6.4.1.2 Récompenses

Le principe de base pour réussir une simulation de jeu est de réaliser ce qui a été décrit par l'entraîneur sans perdre le ballon (pour l'adversaire ou s'il sort du terrain). Selon ce principe nous avons choisi de donner une récompense à l'agent pour chaque objectif réussi et une punition à toute l'équipe si jamais elle perd le ballon. Les agents reçoivent une récompense nulle (ou n'ont pas de récompense) pour les autres situations.

Le Tableau 6.2 montre les valeurs que nous avons utilisées et résume ce que nous venons de dire.

Situation	Récompense	Envoyée à
Réussir un objectif	1	agent
Perdre le ballon ou le mettre hors du terrain	-1	équipe
Autre	0	agent

Tableau 6.2 – Récompenses (*variation de la distance entre l'agent et son objectif normalisé).

Ce sont les éléments de base de l'algorithme de l'apprentissage par renforcement que nous avons utilisé pour le problème de la simulation des situations de jeu. La section suivante présente la façon dont nous avons intégré le transfert de connaissance.

6.4.1.3 Algorithme d'apprentissage et transfert de connaissance

En essayant d'avoir un apprentissage plus rapide et plus fin, nous avons ajouté le transfert de connaissance selon l'algorithme d'optimisation du transfert avec l'évaluation en ligne que nous avons créé et présenté par la Figure 4.8.

Pour pouvoir utiliser le transfert de connaissance nous avons besoin de la connaissance à transférer, et nous l'avons créée après la réunion avec M. SEBBANE. Pour cela nous avons trois éléments de la situation de jeu et nous avons associé une action à chaque possibilité. Nous avons considéré si l'agent a le ballon ou pas, s'il a un objectif et si l'objectif demande la présence du ballon ou pas. Le Tableau 6.3 montre la table de vérité correspondante : les trois premières colonnes sont les entrées déjà mentionnées et la quatrième est l'action associée observée comme sortie.

Avec le ballon	Avec un objectif	Objectif avec le ballon	Action
Faux	Faux	Faux	Se positionner
Faux	Faux	Vrai	Se diriger vers le ballon
Faux	Vrai	Faux	Se diriger vers l'objectif
Faux	Vrai	Vrai	Se diriger vers le ballon
Vrai	Faux	Faux	Faire une passe pour ce qui a besoin du ballon
Vrai	Faux	Vrai	Se positionner en conduisant le ballon
Vrai	Vrai	Faux	Faire une passe pour ce qui a besoin du ballon
Vrai	Vrai	Vrai	Se diriger vers l'objectif avec le ballon

Tableau 6.3 – Table de vérité et l'association d'une action

En utilisant l'algèbre de Boole nous avons simplifié le Tableau 6.3 et nous avons pu obtenir le système de règles de la Figure 6.3 que nous avons utilisé dans notre algorithme d'optimisation du transfert de connaissance.

```

Si ((avec le ballon) ET (objectif sans le ballon))
    Faire une passe pour ce qui a besoin du ballon
Si ((sans le ballon) ET (objectif avec le ballon))
    Se diriger vers le ballon pour le prendre
Si ((sans le ballon ET sans un objectif ET sans le ballon)
    OR (avec le ballon ET sans un objectif ET avec le ballon))
        Se positionner
Si ((sans le ballon ET avec un objectif ET sans le ballon)
    OR (avec le ballon ET avec un objectif ET avec le ballon))
        Se diriger vers l'objectif

```

Figure 6.3 – Système de règles utilisé pour le transfert.

Nous avons opté pour une approche assez simple au moment de la création de la connaissance à transférer pour montrer les avantages de l'optimisation du transfert de connaissance même si la connaissance disponible n'est pas parfaite ou bien sophistiquée.

Un spécialiste du domaine en restant plus de temps sur le problème pourrait sans doute faire un système de règles plus évolué. Mais comme l'objectif de nos premières expérimentations, présentées dans cette thèse, est de montrer les avantages de notre

algorithme de transfert et ne pas de chercher le meilleur résultat possible. Nous avons donc opté pour l'utilisation d'une connaissance que nous pouvons obtenir d'une façon plus simple cependant bien utile pour accélérer le processus d'apprentissage.

6.5 Quel type d'apprentissage multiagent

Tel qu'il a été défini dans les sections précédentes, le cadre de l'apprentissage par renforcement ne porte que sur un seul agent. Lorsque plusieurs agents évoluent dans le même environnement, l'utilisation d'algorithmes d'apprentissage par renforcement pose de nombreux problèmes. En effet, les interactions entre agents ne sont pas prises en compte par ces algorithmes.

Dès lors, de nombreux algorithmes d'apprentissage par renforcement spécifiquement multiagent ont été développés ces dernières années. Ils prennent en compte les interactions, et permettent aux agents de coopérer et de se coordonner les uns avec les autres.

Le principal problème lié à l'apprentissage par renforcement multiagent est l'explosion de la complexité : la plupart des algorithmes disponibles dans la littérature sont exponentiels en fonction du nombre d'agents (Gies et Chaib-Draa, 2006). Nous avons donc choisi de présenter un court état de l'art des techniques d'apprentissage multiagent, ordonnées suivant les critères de complexité. Plus précisément, soit S l'espace d'état et A l'espace d'action d'un seul agent, et soit n le nombre d'agents. Pour chaque technique présentée ci-dessous, nous présentons la taille globale des espaces d'état et d'action à explorer.

Apprentissage de Joint-Action Learners (Claus et Boutilier, 1997). Dans ce cadre, chaque agent dispose d'un espace d'état dans lequel toutes les informations sur l'ensemble des autres agents sont disponibles. Les agents apprennent chacun en tenant compte des actions des autres. Du fait que les agents ne communiquent pas entre eux dans ce modèle, il leur est nécessaire de prédire les actions des autres joueurs, par exemple avec l'algorithme du fictitious play (Gies et Chaib-Draa, 2006). Taille globale des espaces d'état est de $n \times S^n$ et celle des espaces d'action est de $n \times A^n$.

Apprentissage centralisé. Une solution simple consisterait à considérer l'ensemble des agents comme un seul agent (centralisé). Dès lors, les problèmes posés par la méthode précédente (absence de communication entre agents) ne se posent plus, ce qui réduit la complexité globale. Taille globale des espaces d'état est de S^n et celle des espaces d'action est de A^n .

Apprentissage avec couplage faible entre agents. En considérant qu'il y a un couplage faible entre agents, c'est-à-dire que tous les agents n'ont pas besoin de se coordonner entre eux, on réduit encore la complexité de l'apprentissage (Guestrin *et al.*, 2002). Taille des espaces d'état et d'action globale : S^c , A^c , c étant le couplage maximal entre agents.

Agents apprenants indépendants. Une solution encore moins complexe consiste à concevoir des agents qui apprennent individuellement, sans se coordonner. C'est ce que Claus et al (Claus et Boutilier, 1997) ont appelé les « *independant learners* ». Dans le cas où l'environnement garde des traces du passage des autres agents, une coordination peut malgré tout émerger. Taille des espaces d'état et d'action globale : $n \times S$, $c \times A$

Mise en commun des comportements. Pour simplifier encore le problème d'apprentissage, certains ont proposé de mettre en commun le comportement des agents. C'est par exemple ce qui a été fait dans l'algorithme TPOT-RL (Stone et Veloso, 1998). Taille des espaces d'état et d'action globale : $S \times A$

Apprentissage avec agents indépendants. Agents apprenants indépendants. Une solution encore moins complexe consiste à concevoir des agents qui apprennent individuellement, sans se coordonner. C'est ce que Claus et al (Claus et Boutilier, 1997) ont appelé les « *independant learners* ». Dans le cas où l'environnement garde des traces du passage des autres agents, une coordination peut malgré tout émerger. Taille des espaces d'état et d'action globale : $n \times S$, $c \times A$.

Nous nous situons dans ces derniers cas dans la mesure que nos agents apprennent d'une façon indépendante, car il n'y a pas de mécanisme de coordination explicite entre les agents.

6.6 Expérimentations et résultats

Cette section présente les expérimentations et résultats que nous avons obtenus en appliquant l'approche présentée dans le Chapitre 5 et en utilisant ce qui a été décrit dans les sections précédentes.

Pour interagir avec l'entraîneur, réaliser nos expérimentations et voir les résultats, nous avons implémenté un prototype de la plateforme envisagée avec les principales fonctionnalités. Dans cette première démarche, nous avons choisi de traiter la simulation de situations d'attaque. Cela nous permet de réduire la tâche d'implémentation sans révoquer notre approche.

Bien que l'effort d'implémentation ait été considérable et, mérite en soi, qu'on le détaille, nous n'allons présenter dans ce chapitre que la partie la plus pertinente dans le cadre

de cette thèse à savoir le transfert en apprentissage par renforcement. Pour plus d'information au niveau de l'implémentation vous pouvez vous référer à l'Annexe C.

Notre application est composée de 4 modules qui sont nommés : « *Examples* », « *Coach* », « *Learning* » et « *Light, camera, action!* ». Le module *Examples* sert à manipuler (ouvrir, voir, dérouler, créer, modifier, partager, etc.) des exemples de situations de jeu ou des exercices d'une manière similaire aux outils présentés par la section 5.2.2. L'objectif de ce module est de reprendre des situations réelles de jeu provenant de la numérisation d'une séquence vidéo pour ensuite pouvoir les incorporer dans le processus d'apprentissage. La numérisation des séquences vidéo est en train d'être développé par le groupe du LSIS à Marseille (Mavromatis *et al.*, 2003; Mavromatis *et al.*, 2003). Une fois que le processus de numérisation automatique est au point, nous allons probablement l'utiliser dans notre processus d'apprentissage.

Les autres trois modules sont liés à l'approche que nous avons décrite dans la section 6.1. L'entraîneur utilise le module *Coach* pour dire ce qu'il veut simuler, le module *Learning* pour définir quelques paramètres pour les agents et le module *Light, camera, action!* pour voir le résultat. Le module *Light, camera, action!* est très similaire à un lecteur multimédia, avec des contrôles pour démarrer, arrêter, avancer, revenir, etc. Les deux autres modules (*Coach* et *Learning*) seront détaillés dans ce chapitre.

6.6.1 Configuration des expérimentations

Nous avons réalisé des expérimentations avec treize situations de jeu, qui ont été données par M. SEBBANE, tirées de livres d'entraînement ou de sites web spécialisés dans le domaine. Nous avons également une numérisation d'une situation de jeu fournie par le groupe du LSIS à Marseille.

De cette manière nous avons réuni douze situations de jeu qui sont en accord avec le domaine de l'entraînement du football, que nous avons nommé : *3pass*, *4across_top*, *continuous_cross*, *crisscross*, *give_go*, *negative_space*, *run_pass*, *une-deux*, *turn_attack*, *turn_cross*, *x* et *ex_match*.

En suivant les mêmes principes de la méthodologie d'expérimentation utilisée dans le Chapitre 4, nous avons fait 30 expérimentations de 3000 épisodes avec une longueur maximale d'environ 3 minutes. L'apprentissage par renforcement a été fait en utilisant notre algorithme d'optimisation du transfert de connaissance en ligne avec le *Q-Learning* présenté dans le Chapitre 6.

Les paramètres que nous avons utilisés pour l'apprentissage sont : $\gamma = 0,99$; $\lambda = 0,9$; $\alpha = 0,2$ et un $\varepsilon = 0,2$. En ce que concerne l'algorithme d'optimisation du transfert avec l'évaluation en ligne nous avons utilisé $\varphi = 0,5$; $\Delta = 0,2$; $\Delta_{\min} = 0,01$; $eval_step = 10$ et $c = 0,1$. En effet, notre algorithme ne dépend pas trop des paramètres choisis, il va les trouver en fonction de la connaissance de transfert. Nous avons choisi ces paramètres juste pour commencer sans faire aucune considération entre à la connaissance de transfert et la connaissance de l'agent.

Etant donné la taille du problème, nous ne pouvons pas traiter la mémoire des agents (c.-à-d. la fonction Q) de la même manière que nous avons fait pour le *gridworld*. Il n'est donc plus possible d'utiliser un tableau pour stocker ce que l'agent apprend. Ainsi, pour traiter cette problématique, nous avons utilisé une méthode de généralisation par approximation de fonctions intitulée CMAC, cette méthode a été détaillée dans la section 2.6.1.2. Selon ce que nous avons montré dans cette même section nous avons utilisé un CMAC avec 9 quadrillages de 32x32 régions.

Pour les expérimentations nous avons mis des adversaires qui ne font pas de l'apprentissage automatique et qui ont leur comportement codé à travers des scripts. L'implémentation que nous avons faite suit à peu près le système à base de règles de la Figure 6.3 utilisé pour le transfert. Cela va produire des défenseurs qui vont vers leurs objectifs quand ces derniers n'ont pas du ballon pour l'atteindre ou vont, dans le cas contraire, essayer de récupérer le ballon.

L'application de notre algorithme à chaque situation de jeu produit deux courbes qui montrent l'évolution de l'apprentissage de nos agents avec et sans transfert. Ces courbes présentent la moyenne du gain obtenu (axe des ordonnées) sur les 30 expérimentations en fonction du déroulement des épisodes d'apprentissage (axe des abscisses).

Toutes les courbes de ce chapitre vont suivre cette description, c'est-à-dire que dans toutes les courbes nous avons les épisodes sur l'axe des abscisses, et sur l'axe des ordonnées la moyenne du gain obtenu sur 30 expérimentations. La courbe continue (rouge) représente le résultat obtenu en utilisant notre technique de transfert de connaissance et l'autre courbe pointillée (verte) le résultat obtenu avec le *Q-Learning*. Pour des raisons de visibilité nous présentons les courbes avec les 700 premiers car le début de l'apprentissage est la partie la plus pertinente au transfert de connaissance.

La section suivante présente en détail les situations de jeu que nous avons utilisées et les résultats que nous avons obtenus.

6.6.2 Résultats

Avant de présenter les résultats nous introduisons la notation que nous avons utilisée pour dessiner les diagrammes qui vont décrire graphiquement les situations de jeu que nous avons expérimentées. Cette notation est assez utilisée par les entraîneurs, par les livres et par les logiciels du domaine. La Figure 6.4 montre tous les éléments avec la notation que nous avons utilisée.

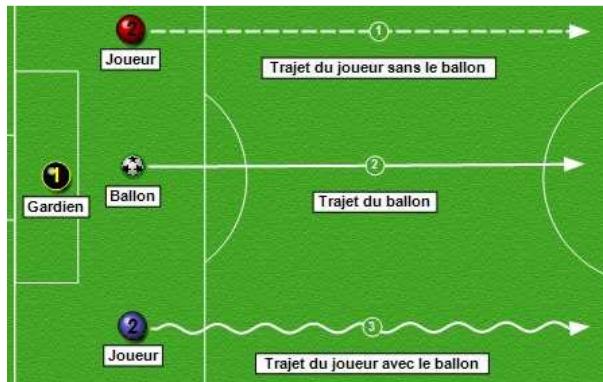


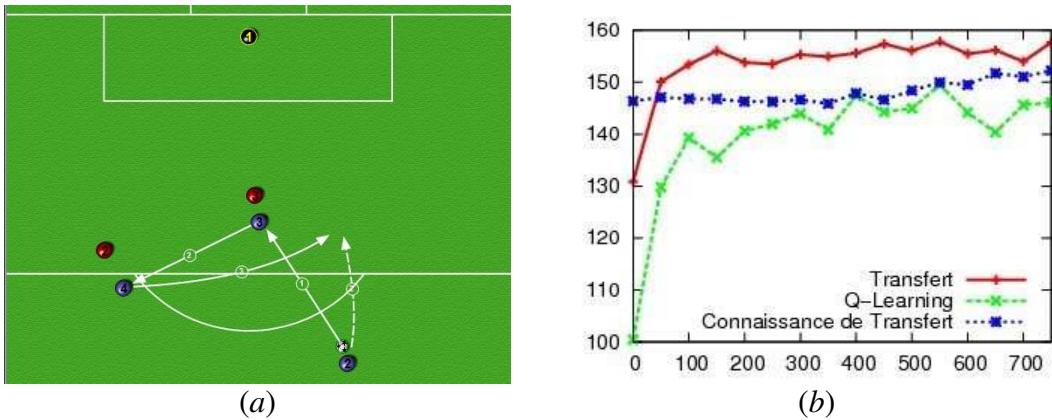
Figure 6.4 – Eléments utilisés dans les diagrammes

Dans la Figure 6.4, nous pouvons voir les joueurs des deux équipes (bleu et rouge), le gardien et le ballon ainsi que les flèches qui indiquent le déplacement du ballon (trait continu) et le déplacement des joueurs sans (trait pointillé) ou avec le ballon (trait continu ondulé).

Une fois la notation comprise nous pouvons passer aux résultats qui seront présentés par des paires de figures : la première montrant le diagramme¹ de la situation de jeu et la seconde indiquant la moyenne du gain obtenu avec et sans l'utilisation de notre algorithme de transfert.

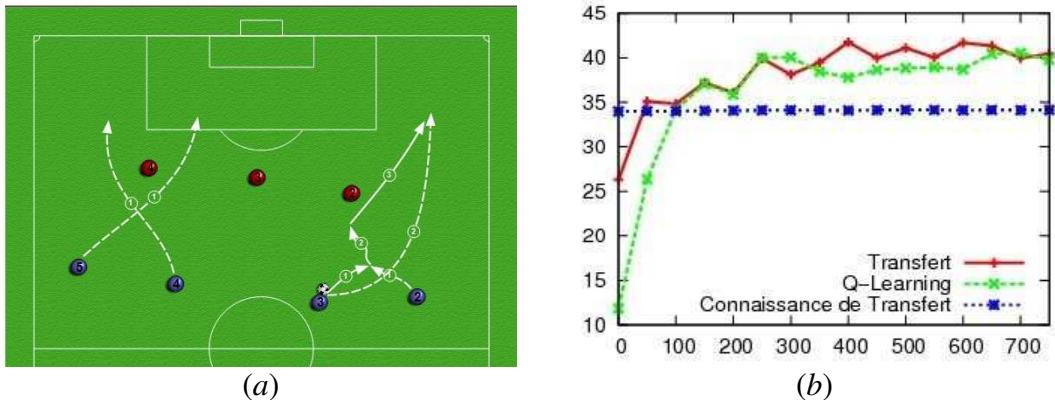
Nous avons préféré illustrer les situations de jeu par des diagrammes synthétiques (dessinés et avec les flèches) plutôt que de présenter la séquence de frames correspondantes dans un souci de simplicité et de cohérence avec la littérature du domaine. Pour plus de détails sur les séquences de frames utilisées vous pouvez vous référer à l'Annexe D.

¹ Dessinés avec le logiciel DataCoach

**Figure 6.5 – Diagramme et courbe d'apprentissage : « 3pass »**

Notre première situation de jeu est présentée par la Figure 6.5(a), elle consiste à faire 3 passes pour placer un joueur en position de tir. Le joueur 2 situé plus à droite commence et finit la situation de jeu avec le ballon, il fait une passe au joueur 3 placé plus au centre et se positionne vers l'avant. Le joueur 3 contrôle le ballon et fait une passe au joueur 4 qui va aussi le contrôler et l'envoyer au joueur 2 qui sera en position pour tirer au but.

La Figure 6.5(b) montre les courbes d'apprentissage avec et sans transfert de connaissance et notre algorithme a été plus performant que le *Q-learning* dès le début.

**Figure 6.6 – Diagramme et courbe d'apprentissage : « 4across_top »**

Le diagramme de la Figure 6.6(a) montre deux croisements. Du côté gauche nous avons un croisement simple entre deux joueurs qui n'ont pas le ballon. Tandis que du côté droit, il y a un croisement entre deux joueurs avec le ballon. Dans le deuxième cas le joueur 3 qui est à gauche du joueur 2 lui passe le ballon et court vers la droite ; le joueur 2 va contrôler le ballon et dribbler vers l'avant pour ensuite faire une passe au joueur 3 qui s'est placé à droite.

Les performances des deux algorithmes montrées par les courbes de la Figure 6.6(b) sont bien équivalentes, cela vient du fait que la connaissance de transfert n'est pas bien

adaptée à cette situation de jeu. Même si la performance de notre algorithme n'est pas meilleure que le *Q-learning* le résultat est intéressant parce qu'il illustre la robustesse de notre algorithme dans le cas où la connaissance de transfert n'est pas appropriée à la situation.

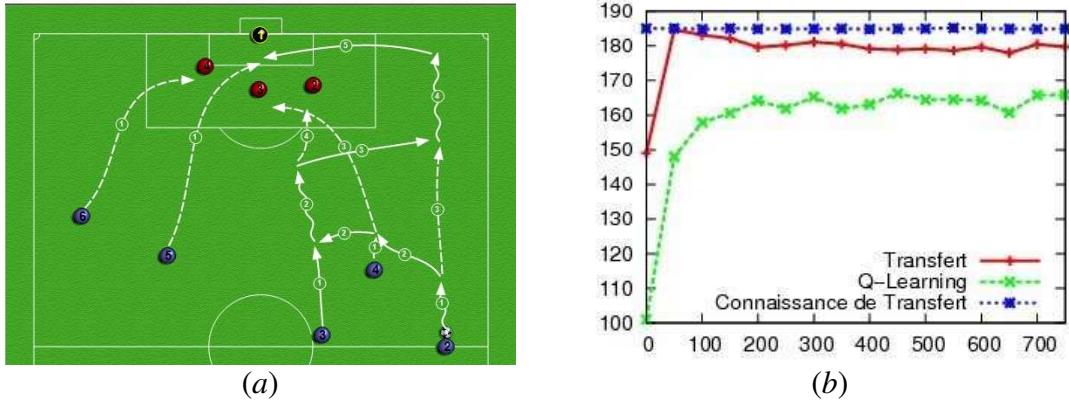


Figure 6.7 – Diagramme et courbe d'apprentissage : « *continuous_cross* »

Dans la Figure 6.7(a) nous avons la préparation et l'exécution d'un centre. Les joueurs 5 et 6 plus à gauche vont se placer dans la surface de réparation tandis que les autres joueurs du côté droit vont s'échanger le ballon pour arriver à se placer à l'entrée de la surface de réparation et placer le joueur 2 sur l'aile droite pour centrer. Les résultats de la Figure 6.7(b) montrent une supériorité de l'utilisation de notre algorithme de transfert.

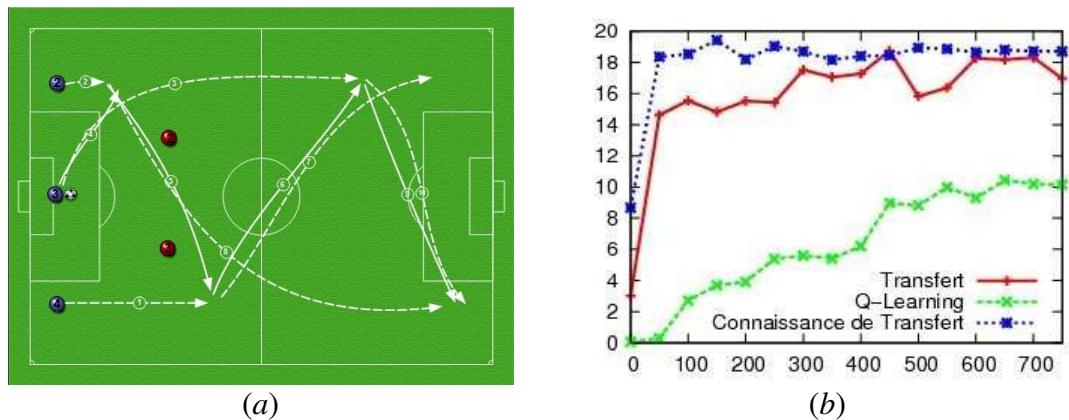


Figure 6.8 – Diagramme et courbe d'apprentissage : « *crisscross* »

La Figure 6.8 illustre de ce que l'on appelle un « *crisscross* », il s'agit d'un croisement avec le ballon qui est utilisé dans les matchs. Dans notre exemple, les 3 joueurs commencent alignés, le joueur qui a le ballon fait une passe au joueur à gauche et poursuit sa course sur la gauche, le joueur qui est à gauche reçoit le ballon fait une passe à droite et part à droite. Et de

cette façon ils peuvent parcourir le terrain en se faisant des passes. Les résultats de la Figure 6.8 (b) montrent une supériorité de l'utilisation de notre algorithme de transfert.

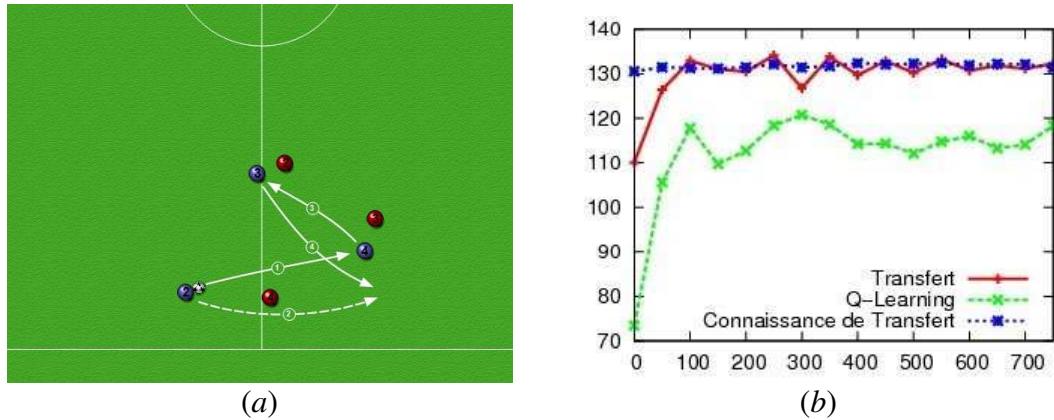


Figure 6.9 – Diagramme et courbe d'apprentissage : « give_go »

Dans la Figure 6.9(a), nous avons une situation proche de celle de la Figure 6.5(a) où il y a l'enchainement de 3 passes. La différence est que cette situation se déroule au milieu du terrain et elle comporte 3 adversaires, mais les résultats du transfert sont toujours supérieurs aux résultats du *Q-learning* (cf. la Figure 6.9(b)).

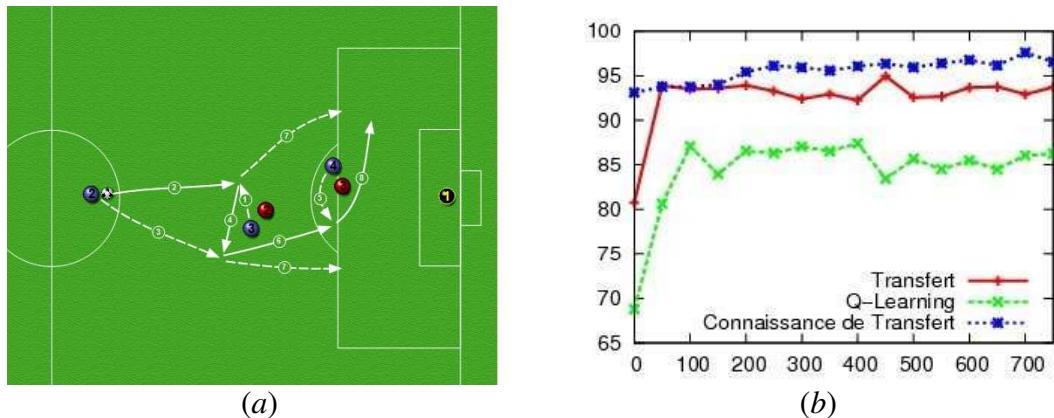


Figure 6.10 – Diagramme et courbe d'apprentissage : « negative_space »

La situation d'attaque de la Figure 6.10(a) montre l'utilisation des espaces vides pour faire avancer le ballon. Le joueur 3 sort de sa position vers la droite pour se démarquer, occuper un espace vide et créer de l'espace avec le défenseur qui va probablement essayer de le suivre. La Figure 6.10(a) montre que le transfert se porte mieux que le *Q-learning* avec une courbe toujours plus haute que le *Q-learning*.

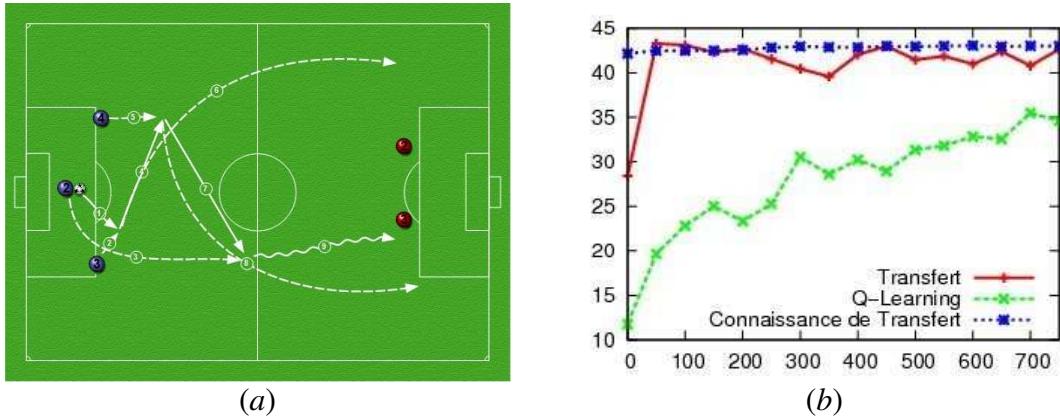


Figure 6.11 – Diagramme et courbe d'apprentissage : « run_pass »

Le départ de la situation de jeu de la Figure 6.11(a) se ressemble à celle de la Figure 6.8(b) où il y a des passes et croisements pour finir avec un joueur qui porte le ballon jusqu'à l'entrée de la surface de réparation et deux autres joueurs que se positionnent sur les ailes. Le transfert obtient des meilleurs résultats que le *Q-learning*, cela est présenté par la Figure 6.11(b).

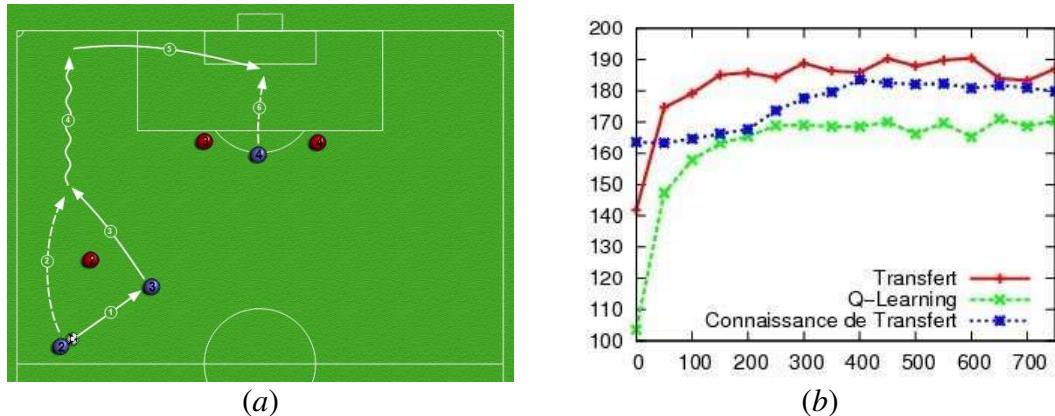


Figure 6.12 – Diagramme et courbe d'apprentissage : « une-deux »

Le une-deux est un mouvement bien connu et assez utilisé dans les matchs. La Figure 6.12(a) montre une attaque qui commence avec un une-deux et qui finit par un centre. Le joueur 2 commence avec le ballon et fait un une-deux avec le joueur 3. Pour ensuite dribler jusqu'au coin et centrer dans la direction du joueur 4. Le gain obtenu par notre algorithme de transfert a été supérieur au gain du *Q-learning* conforme montré dans la Figure 6.12(b).

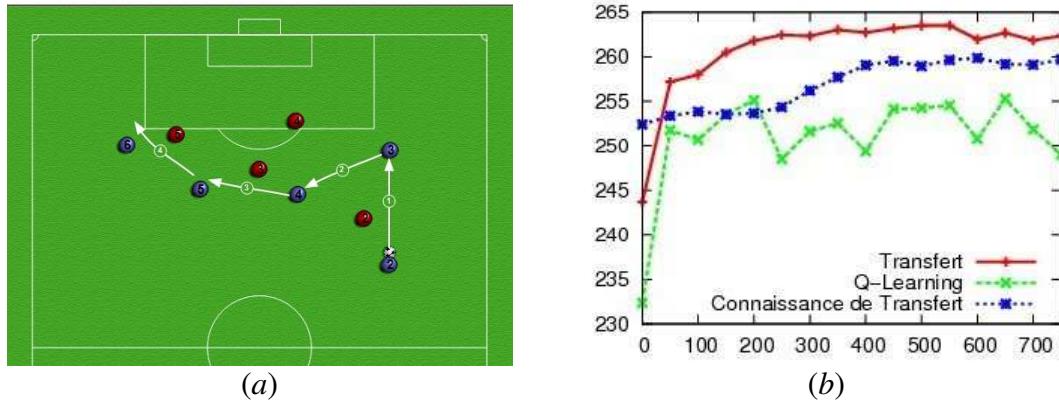


Figure 6.13 – Diagramme et courbe d'apprentissage : « *turn_attack* »

La Figure 6.13(a) montre le tournement d'une attaque de la droite vers la gauche. Le ballon part de la droite vers la gauche à travers d'une séquence de passes réalisées respectivement par les joueurs 2, 3, 4, et 5. La Figure 6.13(b) montre que le transfert a été bien meilleur que le *Q-learning*.

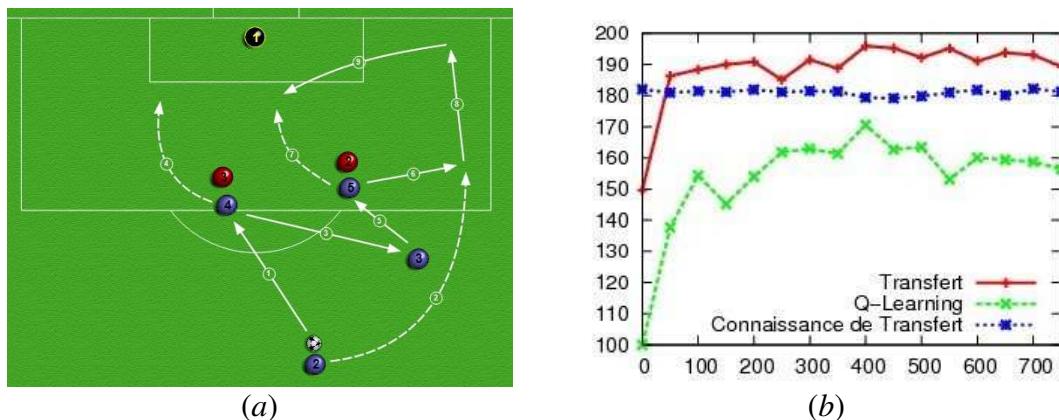


Figure 6.14 – Diagramme et courbe d'apprentissage : « *turn_cross* »

L'attaque de la Figure 6.14(a) montre une séquence de passes pour mettre un joueur en position pour faire un centre. Le ballon fait un mouvement de zigzag entre les joueurs et finit dans les pieds du joueur 2 qui va centrer. La supériorité du transfert est montrée par la Figure 6.14(b).

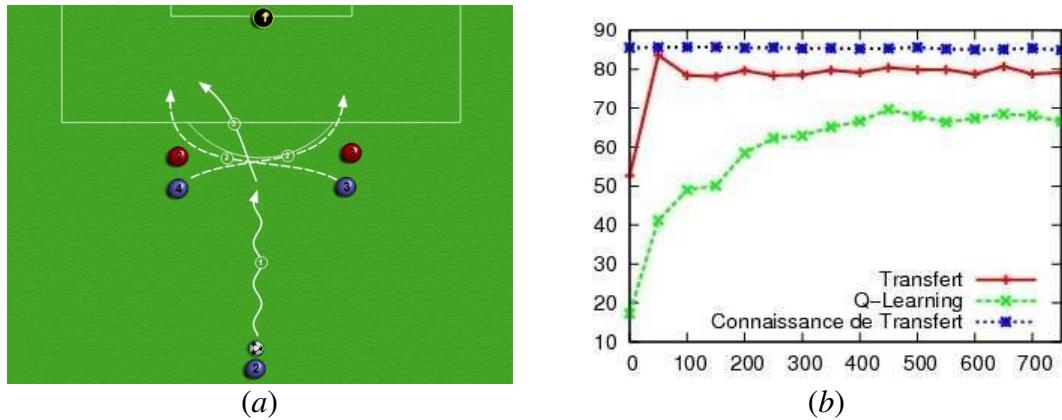


Figure 6.15 – Diagramme et courbe d'apprentissage : « *x* »

Le diagramme de la Figure 6.16(a) indique que le joueur 2 commence avec le ballon et le conduit jusqu'à l'entrée de la surface de réparation, puis les deux autres joueurs font un croisement sans le ballon pour tromper les défenseurs et créer de l'espace. Ensuite le porteur du ballon fait la passe. Les résultats pour cette situation de jeu (Figure 6.16(b)) montrent que le transfert permet un bon départ pour l'apprentissage où les résultats sont bons depuis le début de l'expérimentation.

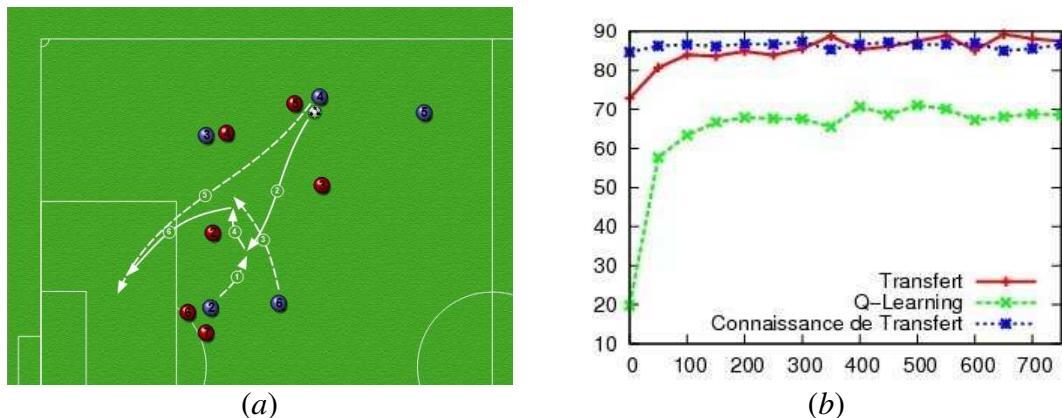


Figure 6.16 – Diagramme et courbe d'apprentissage : « *ex_match* »

Le diagramme de la Figure 6.16(a), montre l'exemple d'une situation de jeu tirée d'un vrai match et qui a été numérisée par le groupe du LSIS de Marseille. Il s'agit donc de déplacements et de passes qui ont été faits par de joueurs professionnels dans une situation réelle.

Le joueur 4 commence l'attaque avec le ballon, il fait une passe au joueur 2, qui se démarque pour le recevoir, et avance vers la surface de réparation. Le joueur 6 fait un croisement de gauche à droite et reçoit le ballon dès qu'il est à droite du porteur de ballon

(joueur 2). Pour finir l'attaque, le joueur 6 fait une passe au joueur 4 qui est dans la surface en bonne position de tir.

La Figure 6.16(b) présente les courbes pour la numérisation utilisée et le transfert s'est montré également plus efficace pour le cas réel.

6.7 Conclusions

Les expérimentations que nous avons faites ont montrées que notre algorithme d'optimisation du transfert de connaissance en ligne est bénéfique pour le processus d'apprentissage.

Même en utilisant une connaissance de transfert qui a été créée de manière très simple, sans disposer d'un système de règles optimisé pour les situations de jeu, l'algorithme de transfert a obtenu les meilleurs résultats. Il a même réussi à s'adapter lorsque la connaissance de transfert n'était pas adaptée à la situation de jeu, comme l'illustre la Figure 6.6 et ce en cessant d'utiliser la connaissance de transfert pour prioriser l'apprentissage.

Les expérimentations que nous avons faites nous montrent donc que notre algorithme est tout à fait applicable à des problèmes réels et aussi complexes que la simulation de situations de jeu. Elles montrent également que son utilisation apporte de bons avantages par rapport au *Q-learning* et qu'il est assez robuste pour s'adapter à la pertinence de la connaissance de transfert par rapport à la situation considérée. La solution que nous avons proposée au problème est donc assurée.

Chapitre 7

Conclusions et perspectives

Nous avons donc étudié et proposé une nouvelle solution pour le problème de la simulation de situations de jeu. Cette solution apporte une nouvelle approche pour le problème en utilisant des agents apprenants autonomes pour réaliser la situation de jeu.

L'utilisation de ce type de technologie nous a permis de simplifier la description de la situation de jeu et l'interaction avec l'utilisateur du système conforme à ce que nous avons présenté dans la section 6.2. Dans notre approche, la description donnée n'a besoin d'exprimer que les points principaux de la situation de jeu et que les agents sont suffisamment autonomes pour inférer la partie manquante afin de réaliser la situation indiquée.

Pour réaliser la situation de jeu donnée, nous avons crée des agents autonomes capables d'apprendre par renforcement la partie manquante et de s'adapter au comportement des adversaires ou à de nouvelles situations de jeu. Avec cette approche, nous avons évité la tâche complexe d'implémenter le comportement des agents avec toutes les réactions qui sont

inhérentes à un joueur. Dans ce cas, nous n'avons pas besoin de prévoir et d'implémenter un comportement pour toutes les situations possibles.

Cependant, l'apprentissage par renforcement a l'inconvénient d'avoir un temps d'apprentissage pour commencer à présenter des bons résultats. Pour réduire ce temps d'apprentissage, nous avons utilisé une technique de transfert de connaissance que nous avons développée (Chapitre 4). Nous avons créé un algorithme générique et robuste, étant donné qu'il est indépendant du format de la connaissance de transfert disponible et qu'il s'adapte à la qualité de cette connaissance de transfert.

La technique de transfert de connaissance en apprentissage par renforcement, que nous avons créée, consiste à utiliser la connaissance disponible d'une manière adaptative. Elle évalue continuellement la connaissance disponible et l'utilise en fonction du résultat de cette évaluation. Cela permet d'utiliser d'autant plus la connaissance disponible quand elle est adaptée au problème ou de l'éviter quand elle est moins pertinente.

Avant d'appliquer notre transfert adaptatif au problème de la simulation de situations de jeu, nous l'avons testé et comparé aux travaux de transfert de connaissance en apprentissage par renforcement existants en utilisant le gridworld. A notre connaissance, cette comparaison est également originale dans la littérature du domaine, car les travaux font souvent référence à ses avantages par rapport à l'apprentissage par renforcement sans le transfert, mais ils ne font pas une comparaison avec une autre méthode de transfert de connaissance.

Une fois que nous avons développé et assuré l'efficacité et performance de notre algorithme, il a été ajouté à notre système de simulation de situations de jeu. L'inclusion de cet algorithme de transfert de connaissance a augmenté la vitesse d'apprentissage des agents et nous avons obtenus de bons résultats. Les résultats obtenus ont confirmé le bon fonctionnement de notre algorithme de transfert sur un problème réel et complexe pour l'apprentissage par renforcement.

De nombreuses directions restent ouvertes pour la simulation de situations de jeu et pour le transfert en apprentissage par renforcement. L'utilisation de plusieurs sources de connaissance est une voie possible pour le transfert de connaissance. Nous pourrons imaginer par exemple une bibliothèque de connaissances avec différents types de connaissance pour des différentes situations. Une analyse théorique de l'algorithme de transfert et un travail plus approfondi au niveau de l'apprentissage par renforcement (p.ex. : MPD factorisés) peuvent être réalisés.

Une autre voie serait d'avoir une version du simulateur totalement adaptative. L'outil qui a été implémenté pourrait être validé avec des professionnels du sport: cela permettrait d'améliorer la version actuelle dans le but de construire une version finale utilisable.

Pour terminer, cette thèse a apporté des résultats originaux et non négligeables pour le transfert de connaissance en apprentissage par renforcement avec la création, expérimentation et comparaison d'un algorithme de transfert adaptatif qui est à la fois générique et robuste, en même temps qu'une nouvelle solution au problème de la simulation de situations de jeu.

Cette solution est doublement originale parce que : d'une part personne n'avait encore traité ce problème avec des techniques d'intelligence artificielle (systèmes multiagent, apprentissage automatique, etc.), et d'autre part, il s'agit de la première application réelle des techniques de transfert de connaissance que nous avons créée.

Bibliographie

(1998). RoboCup-97: Robot Soccer World Cup I. In: RoboCup. Kitano H. ed., Springer, vol. 1395, pp. (Lecture Notes in Computer Science).

Abbeel P., Ng A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, pp. 1. ACM, Banff, Alberta, Canada.

Albus J. S. (1975), A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement and Control, American Soc. of Mechanical Engineers*, 97, pp. 220-227.

Albus J. S. (1981). Brains, Behavior and Robotics. McGraw-Hilp. (BYTE Books).

Andou T. (1998). Andhill-98: A RoboCup Team which Reinforces Positioning with Observation. In: RoboCup. pp. 338-345.

Atkeson C. G., Schaal S. (1997), Robot learning from demonstration. *Proc. 14th International Conference on Machine Learning*, pp. 12--20.

Auer P., Cesa-Bianchi N., Fischer P. (2002), Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47, 2, pp. 235-256.

Bakker P., Kuniyoshi Y. (1996). Robot see, robot do: An overview of robot imitation. In AISB96 Workshop on Learning in Robots and Animals.

Bellman R. E. (1957). Dynamic Programming. Princeton, New Jerseyp. (Princeton University Press).

Bentivegna D. C., Atkeson C. G. (2002). Learning How to Behave from Observing Others. *SAB'02-Workshop on Motor Control in Humans and Robots: on the interplay of real brains and artificial devices*, Edinburgh, UK, August 2002, pp.

Bentivegna D. C., Atkeson C. G., Cheng G. (2002). Humanoid Robot Learning and Game Playing Using PC-Based Vision. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2449-2454.

Bentivegna D. C., Ude A., Atkeson C. G., et al. (2002). Humanoid Robot Learning and Game Playing Using PC-Based Vision. *IROS 2002*, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland, October 2002, pp.

Berry D. A., Fristedt B. (1985). Bandit Problems: Sequential Allocation of Experiments. Chapman and Hall. Londonp.

Bertsekas D. P., Tsitsiklis J. N. (1996). Neuro-Dynamic Programming. Athena Scientific, 512 p.

Bhaskara M. (2007). Automatic shaping and decomposition of reward functions. In ICML, Vol. 227, pp. 601-608. ACM.

Boutilier C. (1999). Sequential Optimality and Coordination in Multiagent Systems. *Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pp. 478-485.

Boyan J. A., Moore A. W. (1995), Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems*, 7, pp. 369–376.

Brafman R. I., Tennenholtz M. (2003), R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3, pp. 213-231.

Champandard A. J. (page mise à jour le January 25th, 2008). An Overview of the AI in Football Games from Cheating to Machine Learning. <http://aigamedev.com/questions/football-ai-cheating-machine-learning>.

Claus C., Boutilier C. (1997). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. *AAAI-97 Workshop on Multiagent Learning*, pp.

Colombetti M., Dorigo, M., Borghi, G. (1996). Robot shaping: The HAMSTER Experiment. *ISRAM'96, Sixth International Symposium on Robotics and Manufacturing*, Montpellier, France, pp.

Cornuéjols A., Miclet L. (2002). Apprentissage artificiel Concepts et algorithmes. Editions Eyrolles, 620 p.

Crites R. H., Barto A. G. (1996), Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems*, 8, pp. 1017-1023.

Crites R. H., Barto A. G. (1998), Elevator Group Control Using Multiple Reinforcement Learning Agents. *Machine Learning*, 33, 2, pp. 235-262.

Dahl F. (1998- 2004). JellyFish Backgammon.

Degris T. (2007). Apprentissage par renforcement dans les processus de décision Markoviens factorisés. PhD Thesis, Université Pierre et Marie Curie, Paris,

Dietterich T. G. (1998), The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 118–126.

Dorigo M., Colombetti M. (1994), Robot shaping: developing autonomous agents through learning. *Artif. Intell.*, **71**, 2, pp. 321-370.

Dorigo M., Colombetti M. (1997). Robot Shaping: An Experiment in Behavior Engineering. MIT Press, 300 p.

Dybsand E., Kirby N., Woodcock S. (2001). AI in Computer Games: AI for Beginners Discussion – Roundtable Handouts. *Game Developers Conference*, pp.

Fernández F., Veloso M. (2006). Probabilistic policy reuse in a reinforcement learning agent. *the fifth international joint conference on Autonomous agents and multiagent systems*, Hakodate, Japan, pp.

Filliat D. (2004). Robotique Mobile. pp. 173, ENSTA-Paris.

Gies O., Chaib-Draa B. (2006), Apprentissage de la coordination multiagent: Une méthode basée sur le Q-learning par jeu adaptatif. *Revue d'intelligence artificielle*, **20**, 2-3, pp. 383-410.

Guestrin C., Lagoudakis M., Parr R. (2002), Coordinated reinforcement learning. *Proceedings of the Nineteenth International Conference on Machine Learning*, pp. 227–234.

Helbing D., Farkas I., Vicsek T. (2000), Simulating Dynamical Features of Escape Panic. *Letters to Nature*, **407**, pp. 487-490.

Hess R. A., Modjtahedzadeh A. (1989). A preview control model of driver steering behavior. *IEEE International Conference on Systems, Man and Cybernetics*, Cambridge, MA, USA, pp. 504-509.2).

Jun K., Sung M., Choi B. (2006). Steering Behavior Model of Visitor NPCs in Virtual Exhibition In: Advances in Artificial Reality and Tele-Existence. Heidelberg S. B. ed., Berlin, vol. 4282/2006, pp. 113-121. (Lecture Notes in Computer Science).

Kaelbling L. P., Littman M. L., Cassandra A. R. (1998), Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, **101**, pp. 99-134.

Kaelbling L. P., Littman M. L., Moore A. W. (1996), Reinforcement Learning: A Survey. *Arxiv preprint cs.AI/9605103*, pp.

Kearns M., Singh S. (1999). Finite-sample convergence rates for Q-learning and indirect algorithms. *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pp. 996-1002.

Kitano H., Asada M., Kuniyoshi Y., *et al.* (1995). RoboCup: The Robot World Cup Initiative. *IJCAI-95 Workshop on Entertainment and AI/ALife*, Montreal, Canada, August 1995, pp. 19-24.

Konidaris G., Barto A. G. (2006). Autonomous shaping: knowledge transfer in reinforcement learning. In ICML, Vol. 148, pp. 489-496. ACM.

Lin L.-J. (1991), Programming robots using reinforcement learning and teaching. *Proceeding of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pp.

Lin L.-J. (1992), Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Mach. Learn.*, **8**, pp. 293--321.

Liu Y., Stone P. (2006). Value-Function-Based Transfer for Reinforcement Learning Using Structure Mapping. In Proceedings of the Twenty-First National Conference on Artificial Intelligence, pp. 415-20.

Mahadevan S. (1997). Grid World. Reinforcement Learning Repository - University of Massachusetts, Amherst.

Mataric M. J. (1994). Reward functions for accelerated learning. In In Proceedings of the Eleventh International Conference on Machine Learning, pp. 181-189. Morgan Kaufmann.

Mataric M. J. (1997), Reinforcement Learning in the Multi-Robot Domain. *Auton. Robots*, **4**, 1, pp. 73-83.

Mavromatis S., Baratgin J., Sequeira J. (2003). Analyzing team sport strategies by means of graphical simulation. *ICISP 2003, International Conference on Image and Signal Processing*, Agadir (Maroc), Juin 2003, pp.

Mavromatis S., Baratgin J., Sequeira J. (2003). Toward the design of a simulator to analyze team sport strategies. *Mirage-2003*, Rocquencourt (France), pp.

Mehta N., Natarajan S., Tadepalli P., *et al.* (2008), Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, **73**, 3, pp. 289-312.

Michie D., Chambers R. A. (1968), BOXES: An experiment in adaptive control, Machine Intelligence 2, E. Eds. Edinburgh: Oliver and Boyd, pp137-152, pp.

Nehaniv C., Dautenhahn K. (1998). Mapping between dissimilar bodies: Affordances and the algebraic foundations of imitation. *European Workshop on Learning Robots*, Edinburgh, pp. 64-72.

Ng A. Y., Harada D., Russell S. J. (1999). Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In ICML, pp. 278-287. Morgan Kaufmann.

Ng A. Y., Russell S. J. (2000). Algorithms for Inverse Reinforcement Learning. In Proceedings of the Seventeenth International Conference on Machine Learning, pp. 663-670. Morgan Kaufmann Publishers Inc.

Nie J., Haykin S. (1999), A dynamic channel assignment policy through Q-learning. *Neural Networks, IEEE Transactions on*, **10**, 6, pp. 1443-1455.

Nie J., Haykin S. (1999), A Q-learning-based dynamic channel assignment technique for mobilecommunication systems. *Vehicular Technology, IEEE Transactions on*, **48**, 5, pp. 1676-1687.

Peterson G. B. (2004), A day of great illumination: B. F. Skinner's discovery of shaping. *Journal of the Experimental Analysis of Behavior*, **82**(3), pp. 317-328.

Precup D., Sutton R. S. (1998), Multi-time models for temporally abstract planning. *Advances in Neural Information Processing Systems*, **10**, pp. 1050-1056.

Price B., Boutilier C. (1999), Implicit imitation in multiagent reinforcement learning. *Proc. 16th International Conf. on Machine Learning*, pp. 325--334.

Price B., Boutilier C. (2000). Imitation and Reinforcement learning in agents with heterogeneous actions. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pp.

Price B., Boutilier C. (2003), Accelerating Reinforcement Learning through Implicit Imitation. *Journal of Artificial Intelligence Research (JAIR) :* **19**, pp. 569-629.

Randløv J., Alstrøm P. (1998). Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In ICML, pp. 463-471. Morgan Kaufmann.

Ratitch B. (2002). Software for RL in C++.

Reynolds C. W. (1999). Steering Behaviors For Autonomous Characters. *Game Developers Conference GDC1999*, San Jose, California, pp.

Robert C. P., Casella G. (2004). Monte Carlo Statistical Methods. Springerp.

Rubinstein R. Y. (1981). Simulation and the Monte Carlo Method. John Wiley & Sons, Inc. New York, NY, USAp.

Russell S. J., Norvig P. (1995). Artificial intelligence : a modern approach. Prentice Hall, Englewood Cliffs, N.J., xxviii, 932 p. (Prentice Hall series in artificial intelligence).

Santamaria J. C., Sutton R. S. (1996). A Standard Interface for Reinforcement Learning Software.

Schaal S. (1997), Learning from Demonstration. *Advances in Neural Information Processing Systems*, **9**, pp. 1040.

Selfridge O. G., Sutton R. S., Barto A. G. (1985). Training and Tracking in Robotics. In IJCAI, pp. 670-672.

Singh S., Bertsekas D. (1997), Reinforcement learning for dynamic channel allocation in cellular telephone systems. *Advances in Neural Information Processing Systems*, **9**, pp. 974-980.

Skinner B. F. (1938). The Behavior of Organisms: An Experimental Analysis. p.

Sondik E. J. (1971). The Optimal Control of Partially Observable Markov Processes. PhD Thesis, Stanford University,

Stone P., McAllester D. (2001). An architecture for action selection in robotic soccer. Montreal, Quebec, Canada, pp. 316-323.

Stone P., Sutton R. S. (2001). Keepaway Soccer: A Machine Learning Testbed. In: RoboCup. pp. 214-223.

Stone P., Veloso M. M. (1998). Team-Partitioned, Opaque-Transition Reinforced Learning. In: RoboCup. pp. 261-272.

Sutton R. S. (1996), Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, **8**, pp. 1038–1044.

Sutton R. S., Barto A. G. (1998). Reinforcement learning : an introduction. MIT Press, Cambridge, Mass., xviii, 322 p. (Adaptive computation and machine learning).

Sutton R. S., Precup D., Singh S. (1999), Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**, 1-2, pp. 181-211.

Taylor M., Whiteson S., Stone P. (2006). Comparing Evolutionary and Temporal Difference Methods for Reinforcement Learning. In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1321-28.

Taylor M. E., Kuhlmann G., Stone P. eds., (2008). Autonomous transfer for reinforcement learning. International Foundation for Autonomous Agents and Multiagent Systems, Estoril, Portugal, 283-290 p. (Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems).

Taylor M. E., Stone P. (2005). Behavior transfer for value-function-based reinforcement learning. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. ACM, The Netherlands.

Taylor M. E., Stone P., Liu Y. (2007), Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. *J. Mach. Learn. Res.*, **8**, pp. 2125-2167.

Taylor M. E., Whiteson S., Stone P. (2007). Transfer via inter-task mappings in policy search reinforcement learning. In Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. ACM, Honolulu, Hawaii.

Tesauro G. (1990), Neurogammon: a neural-network backgammon program. *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 33-39.

Tesauro G. (1992), Practical issues in temporal difference learning. *Machine Learning*, **8**, 3, pp. 257-277.

Tesauro G. (1995), Temporal difference learning and TD-Gammon. *Communications of the ACM*, **38**, 3, pp. 58-68.

Tesauro G. (2002), Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, **134**, 1, pp. 181-199.

Todd J. C., Carroll J. L., Peterson T. S., *et al.* (2001). Memory-guided Exploration in Reinforcement Learning. *INNS-IEEE International Joint Conference on Neural Networks* Washington, DC, pp. 1002-1007.2).

Van Roy B., Bertsekas D. P., Lee Y., *et al.* (1997), A neuro-dynamic programming approach to retailer inventorymanagement. *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, **4**, pp.

Veloso M., Stone P., Bowling M. (1999). Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. *SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, Boston, pp. September 1999.3839).

Watkins C., Dayan P. (1992), Q-learning. *Machine Learning*, **8**, 3, pp. 279-292.

Watkins C. J. (1989). Models of Delayed Reinforcement Learning, Ph. D. thesis, Cambridge University,

Whitehead S. D. (1991). Reinforcement learning for the adaptive control of perception and action, University of Rochester,

Woodcock S. (2000). Game AI: The State of the Industry.

Yu T. (page mise à jour le Behavior Simulation for Autonomous Agents in Crowded Environment. http://www.cs.unc.edu/~taoyu/COMP790-58/final_report.htm.

Zhang W., Dietterich T. G. (1995), A reinforcement learning approach to Job-shop Scheduling. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1114-1120.

Annexes

Annexe A

Algorithme pour perturber une politique

A.1 Description de l'algorithme

```

/* nperturb = nb d'etats a perturber +1 (voir apres si jamais ca commence a
gener, vu que il y a toujours une case de plus pertubée)*/
void pertub(int nperturb,int lissage, int drawFlag) {
    double perturbation[SIZE][SIZE],perturbation2[SIZE][SIZE];
    double num_list[SIZE*SIZE];
    double threshold;

    int x,r,c,l,a;

    for (r=0;r<= (SIZE-1);r++)
        for (c=0;c<= (SIZE-1);c++)
            perturbation[r][c] = choose_random_value();

    /*je lisse ces perturbations*/
    for (l=0; l < lissage; l++) {
        /*je recopie les perturbation dans ma sauvegarde*/
        for (r=0;r<= (SIZE-1);r++)
            for (c=0;c<= (SIZE-1);c++)
                perturbation2[r][c] = perturbation[r][c];

        for (r=0;r<= (SIZE-1);r++)
            for (c=0;c<= (SIZE-1);c++) {
                int nv=2;
                double s=2*perturbation2[r][c];
                if (r>0) {
                    nv++;

```

```

        s+=perturbation2[r-1][c];
    }
    if (r < (SIZE-1)) {
        nv++;
        s+=perturbation2[r+1][c];
    }
    if (c>0) {
        nv++;
        s+=perturbation2[r][c-1];
    }
    if (c < (SIZE-1)) {
        nv++;
        s+=perturbation2[r][c+1];
    }
    perturbation[r][c] = s/nv +
choose_random_value()/1000000.0;
}
/* je fais monter le lac */

for (x=r=0;r<= (SIZE-1);r++)
    for (c=0;c<= (SIZE-1);c++)
        num_list[x++] = perturbation[r][c];

qsort(num_list,SIZE*SIZE,sizeof(double),(void*)comp_nums);
threshold = num_list[nperturb];

if (drawFlag)
    for (r=0;r<= (SIZE-1);r++)
        for (c=0;c<= (SIZE-1);c++) {
            if (perturbation[r][c] <= threshold) {
                STATE g;
                g.row = r;
                g.col = c;
                int start_x,start_y;
                start_x = state_x_location(g);
                start_y = state_y_location(g);

                draw_direction(NORTH,g);
                draw_direction(SOUTH,g);
                draw_direction(EAST,g);
                draw_direction(WEST,g);
            }
        }
/* creation de qtransfert */
for (r=0;r<= (SIZE-1);r++)
    for (c=0;c<= (SIZE-1);c++)
        for (a=NORTH;a<=WEST;a++)
            if (perturbation[r][c] <= threshold) {
                qtransfert[r][c][a] = qvalues
                    [choose_random_int_value(SIZE)]
                    [choose_random_int_value(SIZE)]
                    [choose_random_int_value(4)];
            } else {
                qtransfert[r][c][a] = qvalues[r][c][a];
            }
}

```

Annexe B

Steering behaviors

Les comportements de navigation (traduction de *steering behaviors*) (Reynolds, 1999) constituent un développement du modèle *boids* créé par Craig Reynolds en 1986. Le modèle informatique *boids* propose une manière de simuler des comportements coordonnés que l'on peut observer dans les vols d'oiseaux ou les bancs de poissons. Le modèle original est fondé sur trois types de comportements élémentaires :

- le comportement de séparation utilisé pour les situations où les agents constituant le groupe sont très proches les uns des autres ;
- l'alignement utilisé pour orienter tous les agents du groupe dans une direction commune ;
- le troisième comportement, la cohésion, oriente un agent en direction de la position moyenne des autres agents du groupe.

Une combinaison de ces trois comportements élémentaires permet une simulation réaliste des comportements qui sont ceux d'une population. Le premier exemple d'utilisation de ce modèle-là a été le court-métrage « Stanley and Stella in: Breaking the Ice ». Ce dernier a été créé en 1987 par Symbolics Graphics Division dans le cadre d'une coopération avec Whitney - Demos Production. Le modèle a été également utilisé dans le domaine de la

robotique, de l'intelligence artificielle, de l'éthologie et de la biologie du comportement, des animations par ordinateur, du cinéma et du théâtre interactif, et des jeux vidéo.

L'approche comportementale (Reynolds, 1999) considère qu'un agent bénéficie d'un répertoire de comportements élémentaires pour atteindre tous ses objectifs dans le domaine d'étude considéré. Un comportement est une loi de contrôle rassemblant un ensemble de contraintes, environnementales ou physiques, et permettant à un agent de satisfaire et/ou de maintenir un objectif. Par exemple, le comportement d'Errance Sans Risques pourra être manifesté par un agent désirant se déplacer aléatoirement (but global) tout en évitant les obstacles et les collisions avec les autres agents (contraintes environnementales locales). Un comportement regroupe en quelque sorte la séquence d'actions qu'un agent doit réaliser pour atteindre son objectif, sur la base d'informations issues de ses perceptions.

B.1 Les comportements élémentaires et les actions de base

Nous avons opté pour l'utilisation des *steering behaviors* parce que ces derniers représentent bien le mouvement des joueurs tout en respectant les lois de la mécanique ce qui ajoute du réalisme aux déplacements dans la simulation. Les modèles utilisés ne permettent ni mouvements ni trajectoires incohérentes du point de vue physique. Les comportements de navigation que nous avons sélectionné sont :

- *Seek* (recherche) : l'agent se dirige vers un point donné ;
- *Cohesion* (cohésion) : l'agent s'approche des autres agents qui sont dans son champ de vision ;
- *Separation* (séparation) : c'est le contraire du précédent, l'agent s'éloigner des autres agents qui sont dans son champ de vision ;
- *Unaligned collision avoidance* (éviter les collisions) : une fois qu'une collision probable est détectée, l'agent change sa direction et ainsi sa trajectoire afin de l'éviter ;
- *Containment* (confinement) : avec ce comportement l'agent reste dans une région déterminée.

Les comportements primaires utilisés seront détaillés dans la section B.5 et les actions implémentées, en utilisant ces derniers, dans la section C.5 de ce rapport.

B.2 Modèle de perception

Pour déterminer l'action idoine à effectuer dans une situation donnée, un agent doit être capable de percevoir son environnement, c'est-à-dire de déterminer l'ensemble des objets présents dans son voisinage, de manière à pouvoir réagir aux modifications susceptibles de survenir.

Le modèle de perception illustré par la Figure 7.1 considère qu'un agent perçoit la position et la vitesse de tous les objets situés à l'intérieur d'un cercle centré sur le centre de gravité de l'agent et d'un rayon R donné.

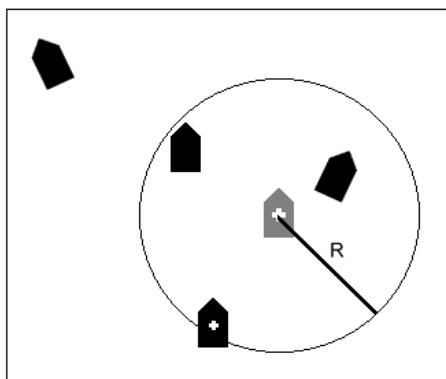


Figure 7.1 – Modèle de perception d'un agent

Ainsi, plus le rayon R est grand et plus le nombre d'objets perçus sera important.

Dans ce schéma, l'agent au centre du cercle perçoit trois agents : un agent est situé à l'intérieur du cercle si son centre de gravité l'est.

B.3 Modèle de locomotion

Ce modèle considère qu'un agent est réduit à son centre de gravité, pour tout ce qui concerne ses déplacements. Toute force susceptible de déplacer l'agent sera donc appliquée uniquement à son centre de gravité, point qui renferme la totalité de sa masse.

Ce modèle permet à la fois de simuler un mouvement assez réaliste et de ne pas nécessiter trop de charge CPU.

Dans ce modèle, l'état interne de navigation d'un agent est défini par son centre de gravité et les informations montrées à la Figure 7.2

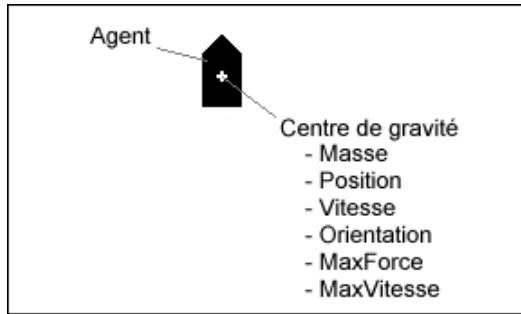


Figure 7.2 – Modèle de locomotion d'un agent

À chaque instant, l'agent se situe dans son environnement à la position indiquée par le vecteur Position, son orientation est définie par son vecteur Orientation et sa vitesse est définie par son vecteur Vitesse.

A chaque pas de temps Δt , l'application d'une force vForce à son centre de gravité déplace l'agent à une nouvelle position égale à Position + Vitesse * Δt , selon le processus suivant :

1. vSteeringForce = truncate(vForce, MaxForce)¹
2. vAcceleration = vSteeringForce / Masse
3. Vitesse = truncate(Vitesse + vAcceleration * Δt , MaxVitesse)
4. Position = Position + Vitesse * Δt

La variable Masse représente la masse de l'agent et permet aux agents lourds d'accélérer moins vite que les agents plus légers pour l'application de la même force de déplacement vForce (lignes 1 et 2). MaxForce symbolise le fait qu'un agent possède une force maximale lui permettant d'exercer une poussée accélératrice. MaxVitesse modélise l'interaction entre l'accélération, produite par la poussée de l'agent, et la décélération, générée par les frottements, de manière à restreindre la vitesse d'un agent à un seuil maximum (ligne 3). Δt est une constante qui représente le laps de temps entre le dernier déplacement de l'agent et le déplacement présent.

B.4 Modèle de collision

Le modèle de collision permet de tester si deux agents sont en collision ou s'ils sont éloignés l'un de l'autre. Le modèle proposé ici, bien qu'approximatif, permet de déterminer rapidement si deux agents se touchent.

¹ truncate(Vecteur v, Scalaire s) – tronque les valeurs des composantes du vecteur v afin que sa norme ne dépasse pas la valeur du scalaire s

Dans ce modèle, chaque agent est supposé occuper dans l'environnement une surface décrite par un cercle centré sur son centre de gravité et possédant un rayon qui l'englobe.

Savoir si deux agents sont en collision revient à tester le signe de la différence entre la distance séparant leurs centres de gravité respectifs d'une part et la somme des rayons des deux cercles englobant d'autre part. Si cette différence est négative ou nulle, les deux agents sont en situation de collision ; si elle est strictement positive, les deux agents sont distants l'un de l'autre.

Ce modèle de collision est illustré par la Figure 7.3 qui montre deux agents en situation de collision sur la gauche, et deux agents distants sur la droite

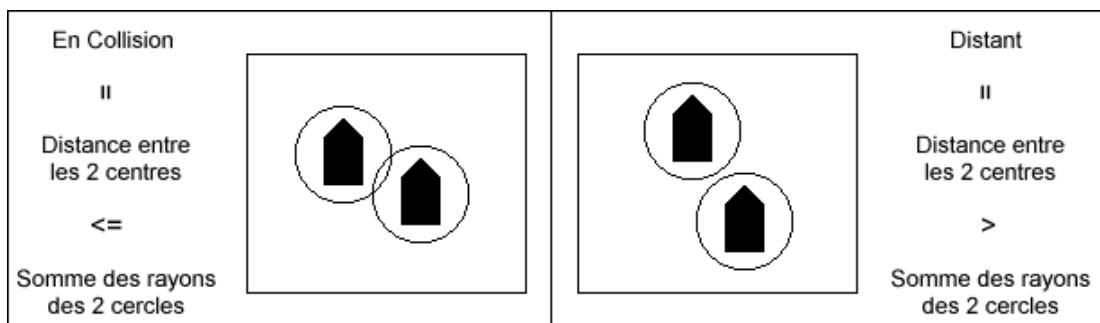


Figure 7.3 – Modèle de collision

B.5 Description des comportements élémentaires

Les comportements élémentaires et les comportements composés sont implémentés au niveau d'un agent sous la forme d'un schéma moteur. Celui-ci se base sur l'état interne de l'agent, comportant son objectif actuel et les informations issues de ses perceptions, afin de déterminer de manière vectorielle l'action motrice de la force vForce à laquelle il sera soumis.

Le comportement d'un agent est ainsi l'attitude observée qu'il exprime face à une certaine situation. Le schéma moteur d'un agent représente l'algorithme implanté à son niveau afin qu'il manifeste un certain type de comportement. Un schéma moteur prend donc en entrée l'ensemble des informations provenant de l'état interne de l'agent et produit en sortie un vecteur de déplacement qui indique la direction et la magnitude de la force auquel sera soumis le centre de gravité de l'agent.

Nous allons présenter dans les sections suivantes les différents schémas moteurs utilisés.

B.5.1 Comportement *seek* (recherche) et *flee* (fuite)

Seek (ou la poursuite d'une cible statique) soumet l'agent à une force telle que son vecteur est dirigé vers la cible. On pourra noter que ce comportement n'est pas celui où l'agent est soumis à une force attractive (comme la gravité) qui produirait un chemin orbital autour du point de cible). Le processus est détaillé dans le paragraphe suivant et illustré par la Figure 7.4. *Desired_velocity* (vitesse souhaitée) est un vecteur qui est porté par la droite joignant la position de l'agent et la cible, qui est dirigé de l'agent vers la cible et dont la norme est, selon le besoin de l'application, *max_vitesse* (vitesse maximale) ou la vitesse actuelle. Le vecteur qui donne la direction de la force (*steering*) est calculé à partir de la différence entre le vecteur *desired_velocity* et le vecteur *current_velocity* (vitesse actuelle).

```
desired_velocity = normalize1 (position - target) *
max_vitesse
steering = desired_velocity - velocity
```

Si jamais un agent continue le comportement *seek*, il traversera la cible, tournera et traversera à nouveau et à nouveau, en produisant un mouvement très semblable au mouvement d'un insecte autour d'une lampe.

Le comportement de fuite (*flee*) est simplement l'inverse du comportement de recherche. Dans ce cas, il faut toujours orienter l'agent de manière à ce que sa vitesse soit radialement alignée avec la cible, mais cette fois-ci dans la direction opposée.

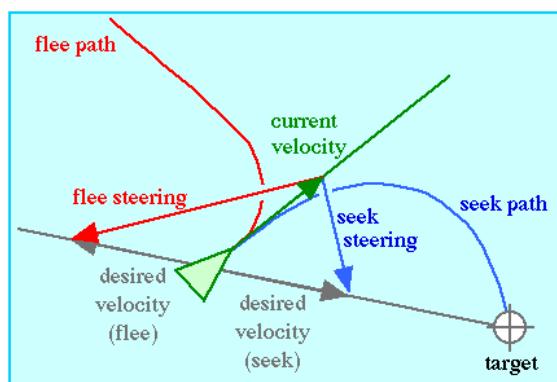


Figure 7.4 – Comportement *seek* et *flee*

B.5.2 Comportement *pursuit* (poursuite) et *evasion* (évasion)

¹ normalize(Vecteur v) – renvoi un vecteur qui a la même direction de v mais une norme égal à 1

Pursuit est comportement qui est semblable à *seek*, sauf que la cible (proie) est un autre agent mobile (cible mobile). Pour réaliser une poursuite, on fait d'abord une première estimation d'une probable position de la proie, puis on prend cette position-là comme cible pour le comportement *seek*. Le secret de la poursuite repose sur la bonne prévision de la future position de la cible.

Evasion est analogue à *pursuit*, sauf que *flee* est employée à la place du comportement *seek*.

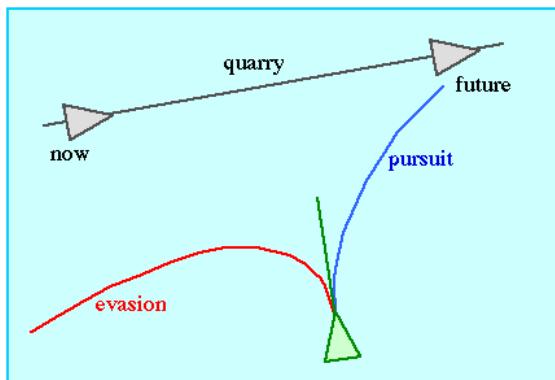


Figure 7.5 – Comportement de poursuite (*pursuit*) et évasion (*evasion*)

B.5.3 Separation (séparation)

Le comportement *separation* donne à un agent la capacité de maintenir une certaine distance entre lui et les autres agents qui lui sont proches. Pour chaque voisin, une force répulsive est calculée en soustrayant les positions des voisins de la position de l'agent. Ces forces répulsives sont additionnées pour chaque agent voisin afin de produire la force résultante. Ce comportement est illustré par la Figure 7.6.

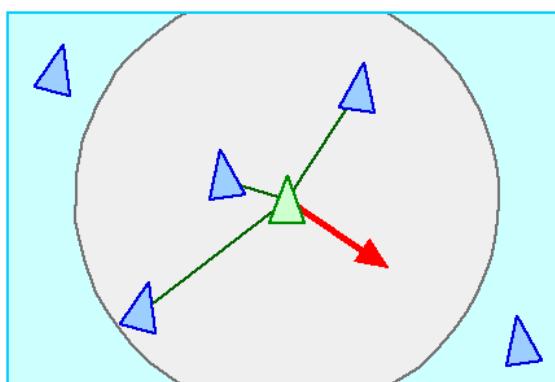


Figure 7.6 – Séparation

B.5.4 Cohesion (cohésion)

Le comportement de navigation intitulé *cohesion* donne à un agent la capacité d'adhérer (s'approcher et constituer un groupe) à d'autres agents voisins (voir la Figure 7.7). On peut trouver la force de cohésion en utilisant « la position moyenne » (centre de gravité) des agents voisins. Ensuite, on applique une force en direction de cette position moyenne, en faisant la soustraction entre la position moyenne et la position de l'agent (comme dans le modèle original de *boids*), en utilisant la position trouvée comme cible pour le comportement *seek*.

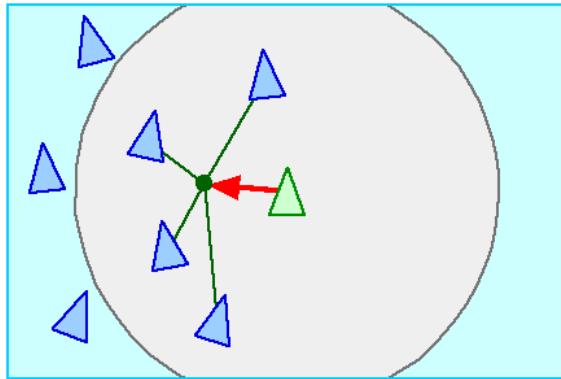


Figure 7.7 – Cohésion

B.5.5 Unaligned collision avoidance (évitement de collisions)

Le comportement *unaligned collision avoidance* essaye d'éviter les collisions entre les agents. Pour le faire, il faut prévoir les collisions potentielles (comme indiqué par la Figure 7.3) et changer la direction et la vitesse afin de les empêcher. Une fois qu'une collision probable est déterminée l'agent s'oriente pour l'éviter, il peut également accélérer ou ralentir pour éviter le point de collision. Sur la Figure 7.8 l'agent qui vient de la droite vers la gauche décide de ralentir et se tourner vers la gauche, tandis que l'autre agent accélère et tourne vers la gauche.

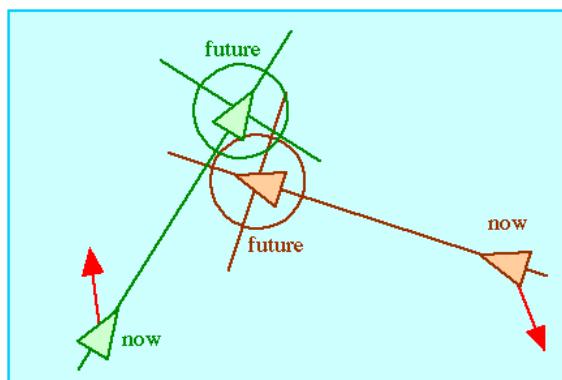


Figure 7.8 – Action d'éviter la collision

B.5.6 *Containment* (confinement)

Le comportement *containment* faire l'agent se déplacer dans une région ou faire l'agent éviter une région. Nous avons comme exemples de cela : des poissons qui nagent dans un aquarium et des joueurs de hockey qui patinent dans une patinoire. Pour l'implémenter, il faut d'abord prévoir la position future de notre agent (trois points sont considérés pour améliorer le comportement : un frontal et deux latéraux). Si jamais cette position est située à intérieur de la région permise, on n'a rien à faire. Dans le cas contraire, il faut l'orienter vers la région permise. Nous pouvons le faire en utilisant le comportement *seek* avec un point à l'intérieur de la région (la projection de la future position vers la normale de la surface d'obstacle par exemple).

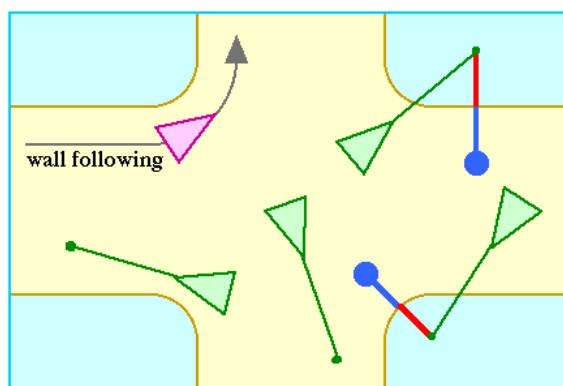


Figure 7.9 – Confinement

Annexe C

Détails d'implémentation du système

Pour interagir avec l'entraîneur, réaliser nos expérimentations et voir les résultats, nous avons implémenté un prototype de la plateforme envisagée avec les principales fonctionnalités. Notre application est composée de 4 modules principaux qui seront décrits prochainement. Une copie d'écran du système est présentée par la Figure 7.10 donne une idée visuelle de ce que nous avons implémenté.

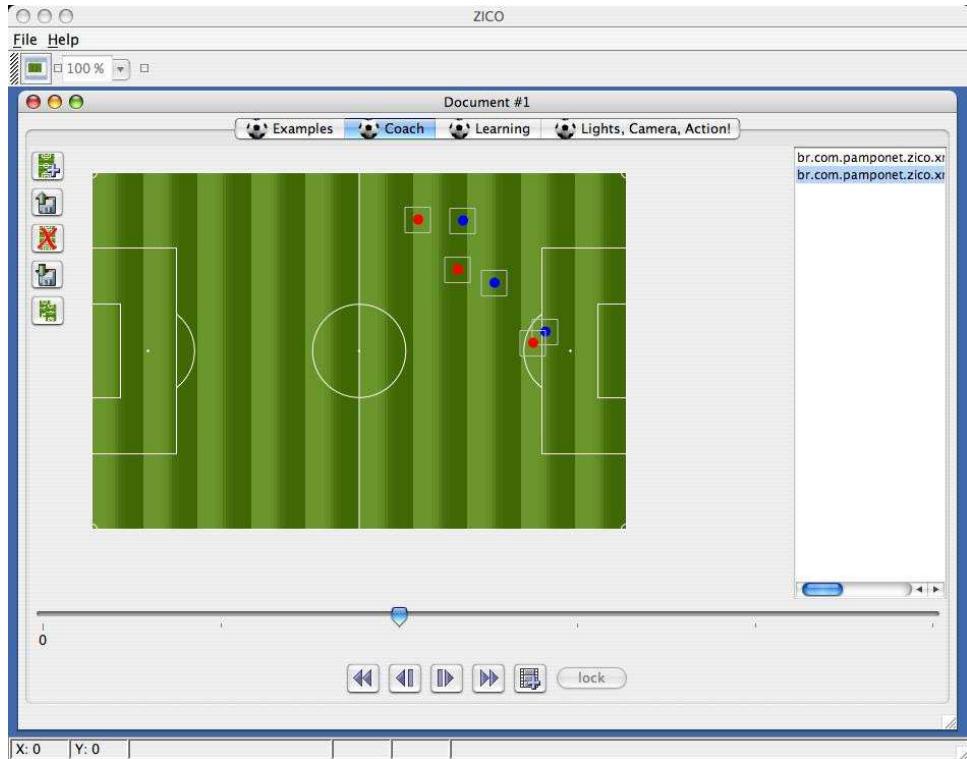


Figure 7.10 – Copie d'écran du prototype

En faisant une brève description des éléments visuels :

- nous avons quatre onglets pour faire la sélection du module souhaité ;
- du côté gauche nous avons les buttons pour créer, ouvrir, supprimer, sauvegarder et dupliquer l'élément considérer par le module ;
- à droite nous avons une liste des éléments qui sont en mémoire ;
- en bas nous avons le panneau de contrôle où nous pouvons manipuler la séquence de situations sur le terrain comme dans un magnétoscope en utilisant les buttons et le *slider* ;
- enfin, nous avons au milieu la représentation graphique du terrain, où au-delà de visualiser l'utilisateur peut modifier ce qu'il voir à la souris (soit les images des exemples, soit le schéma tactique).

C.1 Le module « Examples »

Vu que nous voulons travailler avec des séquences de matches numérisées, nous avons créé un module qui est responsable pour le traitement de toute cette partie.

Pour cela nous avons élaboré une représentation pour les séquences en utilisant XML (*Extensible Markup Language*, ou Langage Extensible de Balisage) et aussi toute la

vérification de la syntaxe à partir du DTD (*Document Type Definition*, ou Définition de Type de Document).

Avec le format du fichier défini, nous avons donc projeté et implémenté toute la partie de traitement de ce fichier. Et également un protocole de création et modification intuitif du point de vue d'utilisateur, vu qu'il n'a d'autre chose à faire que déplacer les éléments du terrain (joueurs, ballon, etc.) à la souris.

C.2 Le module « Coach »

Avant de commencer la description du module *coach* nous allons montrer comment nous avons défini et implémenté un schéma tactique. À partir de la notre connaissance du domaine du football et des logiciels de la section 5.2.2, nous pouvons penser à un schéma tactique comme une séquence d'images (*frames*) constituée de points ou zones du terrain qui sont attribués à tous les joueurs. C'est-à-dire à des moments précis on détermine où (points ou zones du terrain) et comment (avec/sans le ballon) les joueurs doivent se placer. Ces points/zones ont les paramètres : largeur, longueur, position du centre, indication de que le joueur est obligé d'arriver avec le ballon ou pas, et indication de que le joueur doit prendre en compte cette zone ou pas.

Définition 7.1. *Schéma tactique* est une séquence d'au moins deux images (*frames*), où chaque frame est constitué de points ou zones du terrain qui sont rapportés aux joueurs.

Avec la définition d'un schéma tactique à la main nous avons suivi un chemin comme ce qui a été mentionné dans la section précédente. Nous avons créé une représentation XML pour un schéma tactique et le mécanisme d'analyse et validation de la syntaxe en utilisant un DTD qui a été créé avec ce motif-là. Ensuite nous avons fait la partie d'interaction entre le système et l'utilisateur toujours en gardant la méthode intuitive de créer le schéma, ajouter les *frames* et placer les points et zones sur le terrain à la souris.

C.3 Le module « Learning »

Ce module s'en charge de la mise en place des algorithmes d'apprentissage que nous avons implémenté. Du point de vue d'utilisateur, il s'agit d'une interface graphique pour paramétrier l'apprentissage (construction interne du fichier de configuration) et démarrer le processus.

Une chose qu'il faut signaler est la librairie d'apprentissage par renforcement que nous avons implémenté et qu'il n'y avait pas avant, surtout en Java. Le diagramme de classes est présenté par la Figure 7.11.

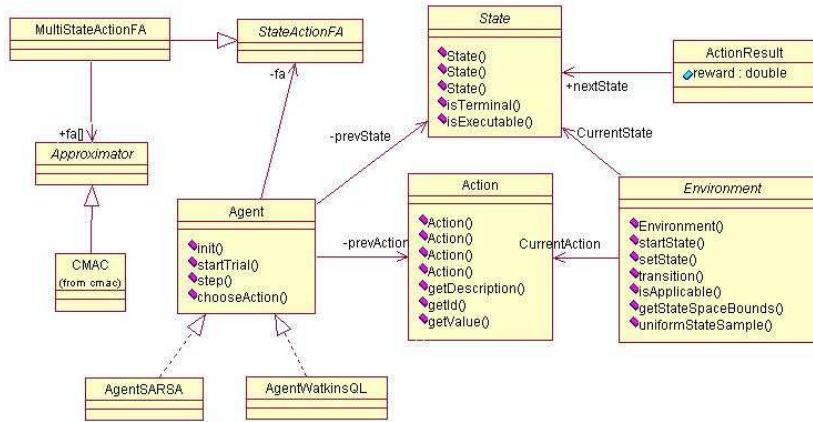


Figure 7.11 – Diagramme de classe du paquet RL

Nous avons les classes **Environment**, **Agent**, **Action** et **State** comme décrit par (Santamaria et Sutton, 1996), mais avec l'implémentation de la structure des mémoires de (Ratitch, 2002) où on peut utiliser n'importe laquelle technique (tableau, réseau de neurones, CMAC, etc.) pour implémenter la mémoire (ce qu'ils ont appris) des agents. L'utilisation de la librairie est très simple, il faut juste dériver quelques classes pour les adapter au problème et profiter des agents et des mémoires déjà implémentés (dans notre cas : SARSA, QL et CMAC).

C.4 Le module « Lights, camera, action! »

Ce dernier module fait essentiellement l'animation des schémas, celui qui permet la visualisation du déploiement des schémas tactiques donnés par l'entraineur.

Le principe de la simulation est très simple et le diagramme des classes principales est présenté par la Figure 7.12. Pour représenter notre monde, nous avons la classe **World** qui a comme attributs le terrain (**SoccerField**) et une liste d'éléments (**WorldObjectSet**) qui appartiennent au monde. Après la création et initialisation du monde et des objets, notre simulation peut être résumée à une boucle principale qui ne s'arrête qu'à la fin de la simulation. Cette boucle peut être divisée en trois parties :

- raisonnement des objets ;
- mis à jour des objets ;

- affichage du monde.

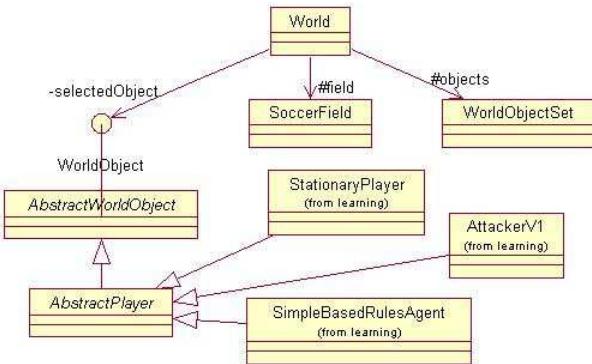


Figure 7.12 – Les classes principales de la simulation

C.5 Implémentation des joueurs

En utilisant comme base les comportements de navigations primaires cités antérieurement (Annexe B), nous avons créé les actions suivantes :

- Se déplacer vers un point : nous avons réalisé cette action-là en combinant les comportements *seek* et *unaligned collision avoidance* afin d'aller vers un point en évitant les collisions ;
- Se déplacer en groupe vers un point : cette fois nous avons ajouté les comportements *cohesion* aux deux autres présents dans l'action antérieure, cette action produit une manière à rester proche du groupe considéré ;
- Se positionner : inspiré de l'action se positionner de (Veloso *et al.*, 1999), nous combinons les comportements de *cohesion*, *separation* et *unaligned collision avoidance*. Cette action va déplacer l'agent vers une zone stratégique qui est au même temps proche de ses compagnons et loin de ses adversaires, toujours en évitant les collisions ;
- Marquer un adversaire : le joueur utilise une heuristique pour déterminer un point proche d'un adversaire qui soit entre le but et le ballon. Ensuite il se dirige vers ce point-là comme dans les cas de l'action se déplacer vers un point ;
- Intercepter le ballon : en utilisant l'algorithme de calcul numérique (Stone et Sutton, 2001) pour trouver le temps pour intercepter le ballon, avec cette information-là l'agent peut savoir vers où il doit y aller pour attraper le ballon.

Il y a d'autres actions également importantes, mais qui n'utilisent pas les comportements de navigation, on peut lister :

- Faire une passe : tir le ballon vers un agent en considérant sa position et vitesse ;
- Prendre le contrôle du ballon : essai de contrôler le ballon, mais le succès dépend des conditions de l'environnement.

C.6 Les actions des joueurs par méthodes

Pour donner une idée des méthodes qui implémentent les actions citées, quelques-unes seront détaillées dans les sous-sections suivantes, sans oublier de détailler ses paramètres. Il faut noter que toutes les méthodes qui traitent le déplacement utilisent le comportement *unaligned collision avoidance* pour éviter les collisions.

C.6.1 La méthode `toMove`

La méthode `toMove` a trois variantes et donc trois signatures différentes :

```
toMove(Point3f p)
      toMove(Point3f p, float seekWeight, float cohesionWeight,
float maxDistance, float cosMaxAngle)
      toMove(Point3f p, float seekWeight, float cohesionWeight, int
K)
```

Le premier est le déplacement simple sans prendre en compte les autres agents, il utilise le comportement *seek* pour aller vers un point donné comme paramètre. Les deux derniers s'agitent de la somme des comportements *seek*, *separation* (relatif aux adversaires) et *cohesion* (relatif aux compagnons). Différemment de la première, ces méthodes prennent en compte les autres agents, cela est le motif d'appeler ce type d'action de se déplacer en groupe vers un point. La Figure 7.13 montre comme la trajectoire du joueur se modifie en fonction des autres.

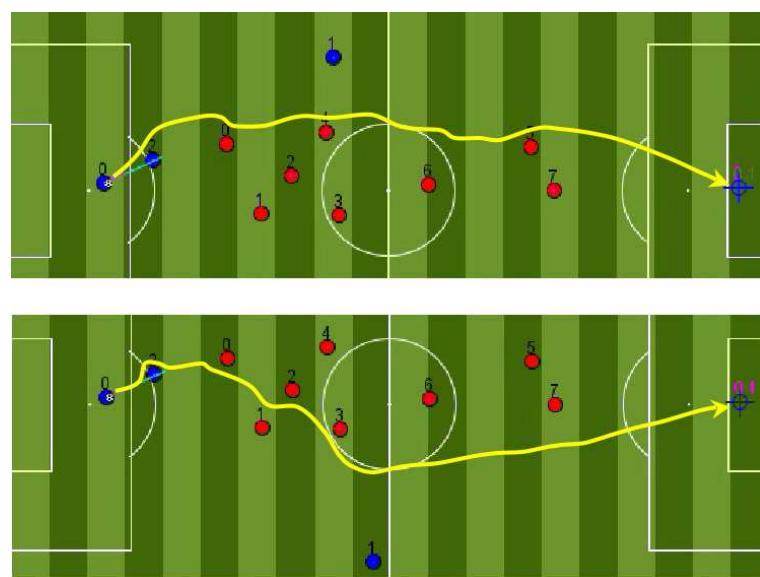


Figure 7.13 – Déplacement en considérant les autres

Les paramètres `seekWeight` et `cohesionWeight` varient entre 0 et 1 inclus et déterminent l'importance ou le poids des comportements *seek* et *cohesion*. Les autres paramètres montrent la différence entre les deux dernières méthodes, cette différence a un rapport avec le groupe qui est pris en compte. La méthode `toMove 2` prend en compte tous les agents qui sont à une distance maximale déterminée par `maxDistance` et dans un angle de vision déterminé par `cosMaxAngle`. La méthode `toMove 3` prend en compte les `K` compagnons plus proches sans considérer la distance ou l'angle de vision.

C.6.2 La méthode `toPosition`

Cette méthode déplace le joueur vers une position qui est au même temps proche des compagnons considérés (en utilisant le comportement *cohesion*) et loin des adversaires considérés (en utilisant le comportement *separation*). Les signatures des méthodes qui implémentent l'action sont :

```
toPosition(float separationWeight, float cohesionWeight, float maxDistance, float cosMaxAngle)
toPosition(float separationWeight, float cohesionWeight, int K)
```

Les paramètres `separationWeight` et `cohesionWeight` déterminent les poids pour les comportements utilisés. Et les groupes de joueurs considérés sont déterminés par les autres paramètres de la même façon que pour la méthode `toMove`. Un exemple de l'action de se positionner est présenté par la Figure 7.14.

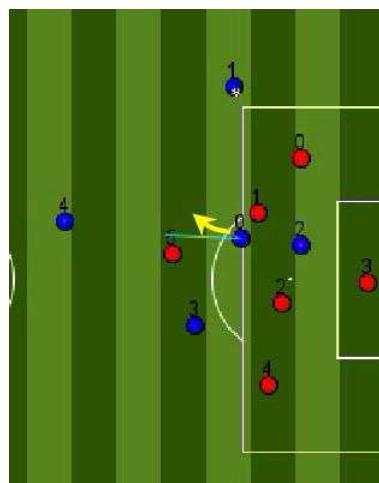


Figure 7.14 – Se positionner

C.6.3 La méthode `toPass`

La méthode de signature `toPass(SimpleVehicle player)` calcule force que le joueur doit appliquer au ballon en considérant la position et vitesse du joueur `player` qui est donné comme paramètre. La force est un vecteur qui a une direction et une norme qui varient en fonction de la distance et direction d'où il faut faire la passe.

Annexe D

Correspondance entre les diagrammes et les descriptions utilisées

Nous présentons dans cette annexe la séquence de frames utilisée pour représenter chaque situation de jeu que nous avons employée dans nos expérimentations

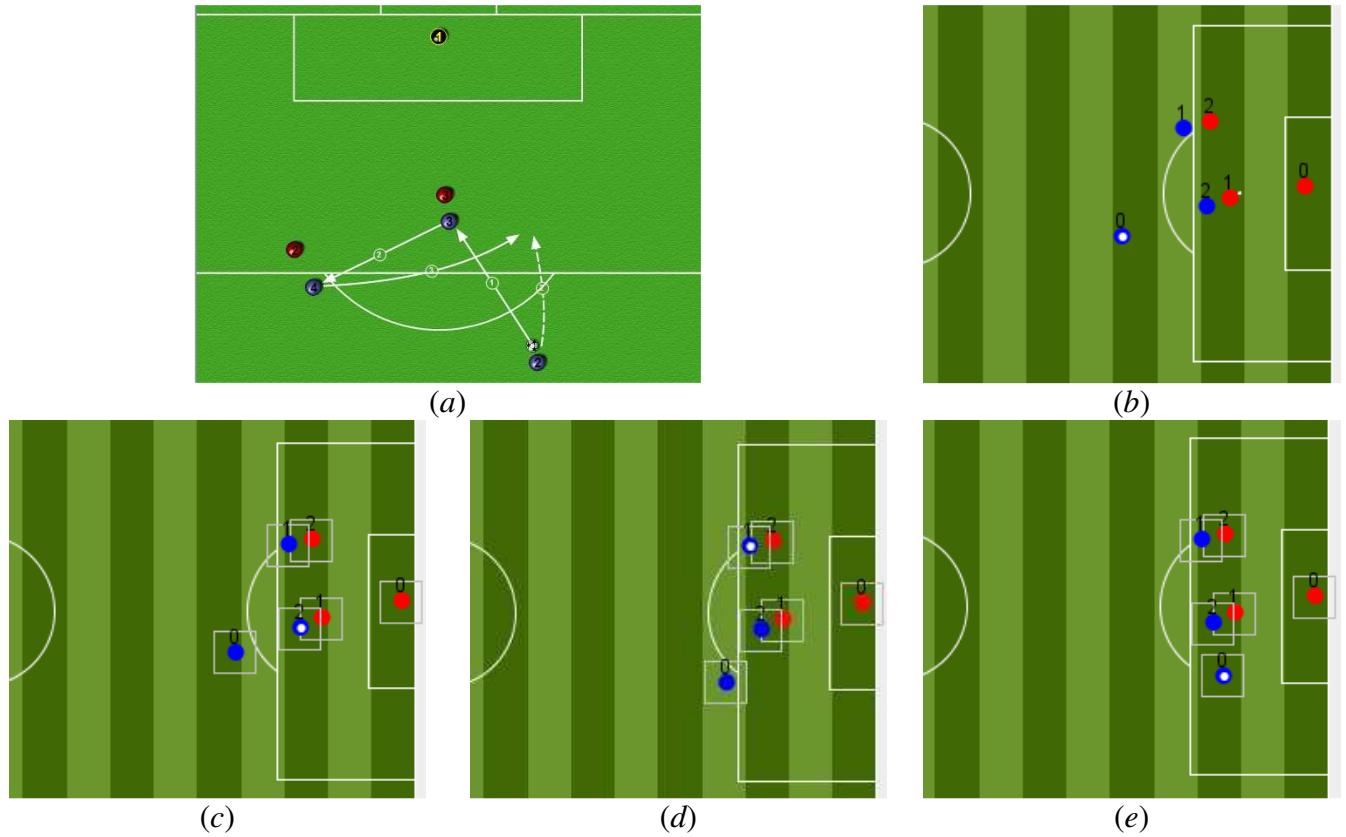


Figure 7.15 – Diagramme et séquence de frames : « 3pass »

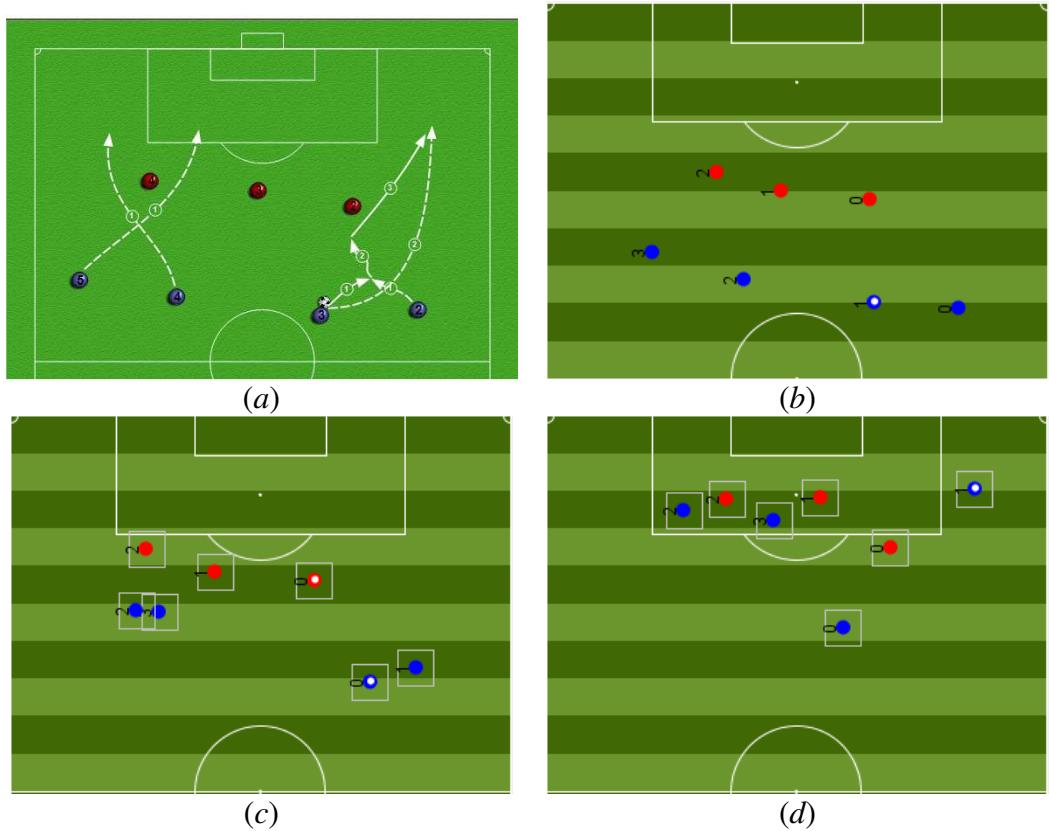
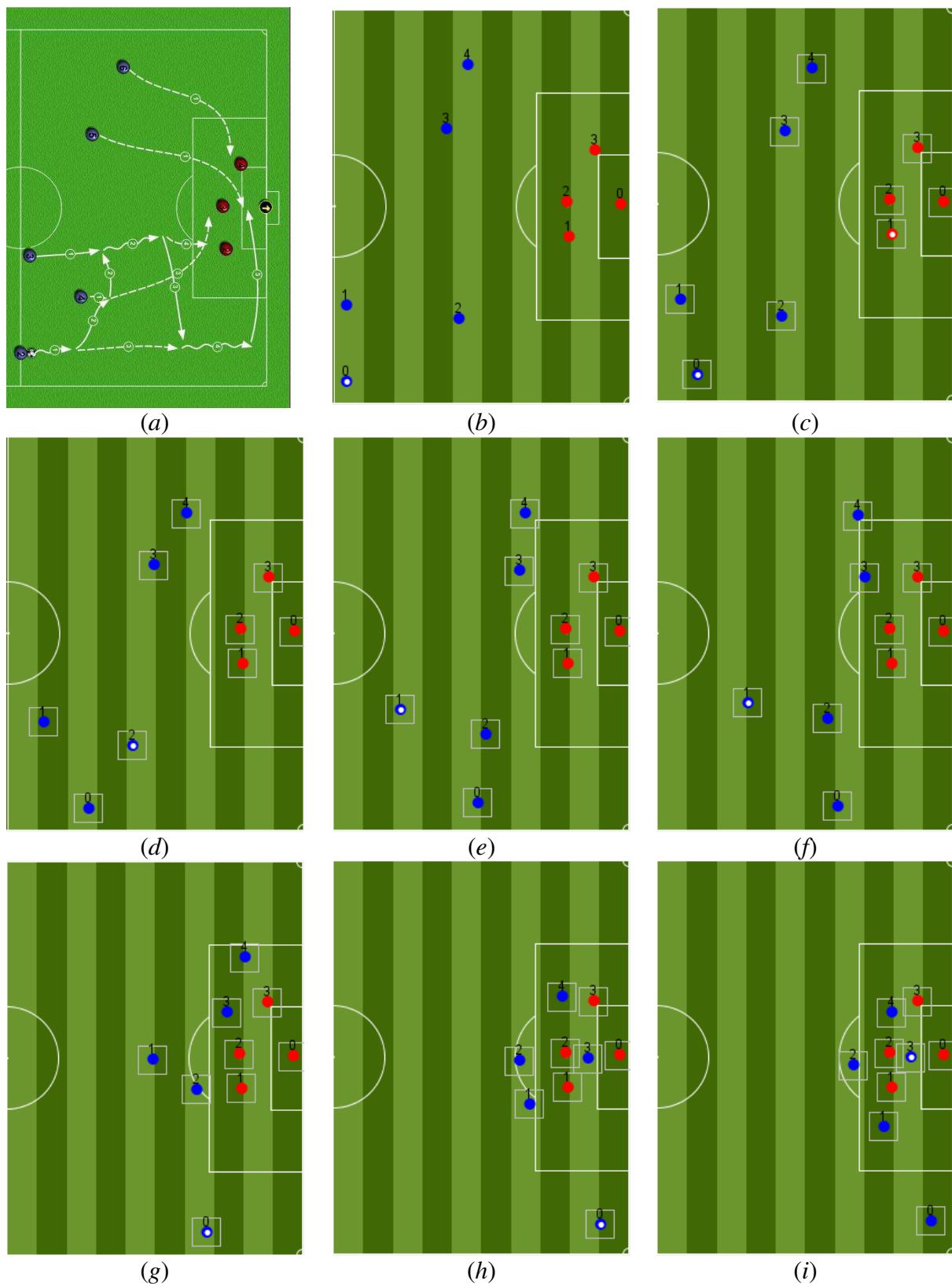


Figure 7.16 – Diagramme et séquence de frames : « 4across_top »

Figure 7.17 – Diagramme et séquence de frames : « *continuous_cross* »

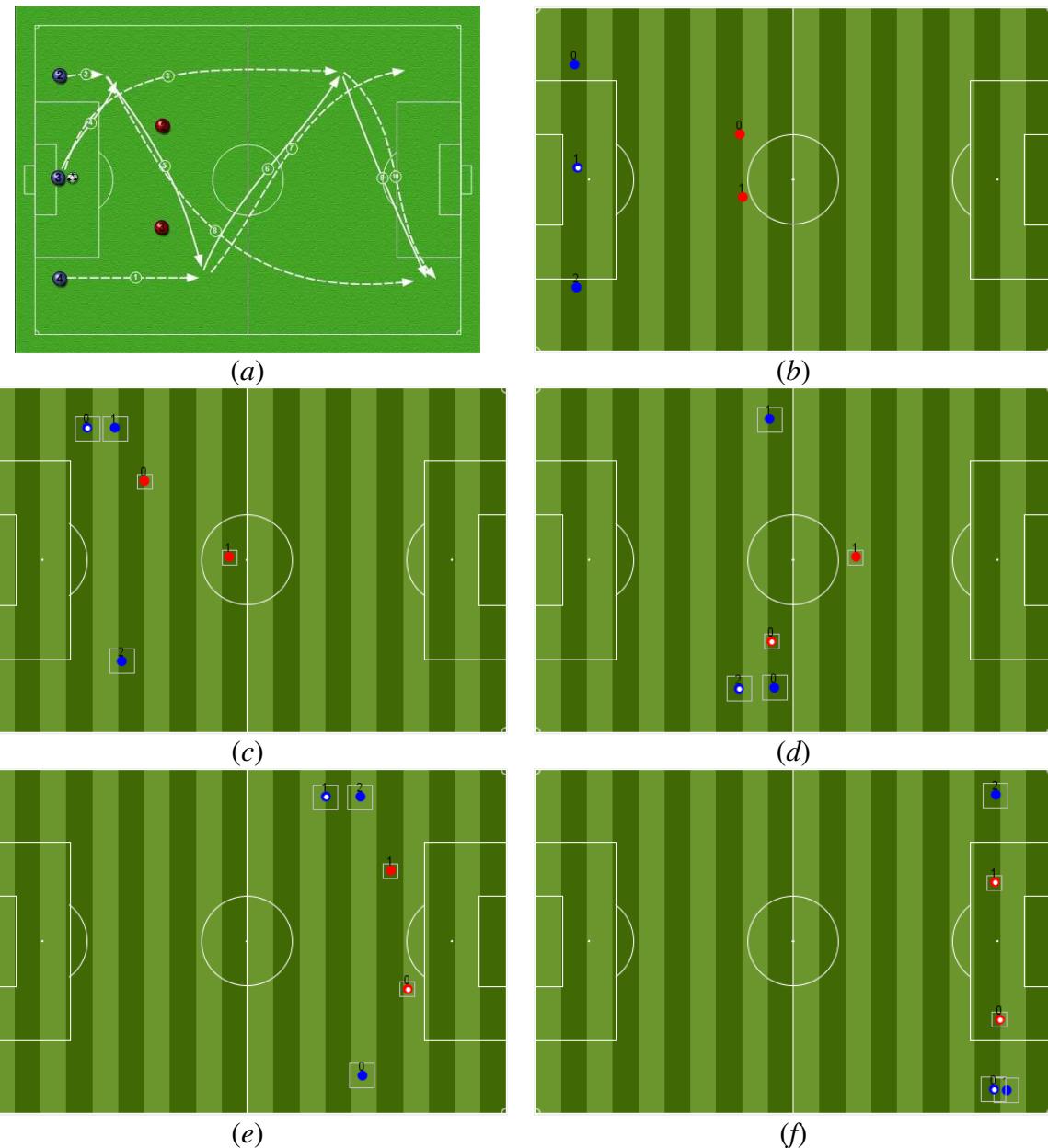


Figure 7.18 – Diagramme et séquence de frames : « crisscross »

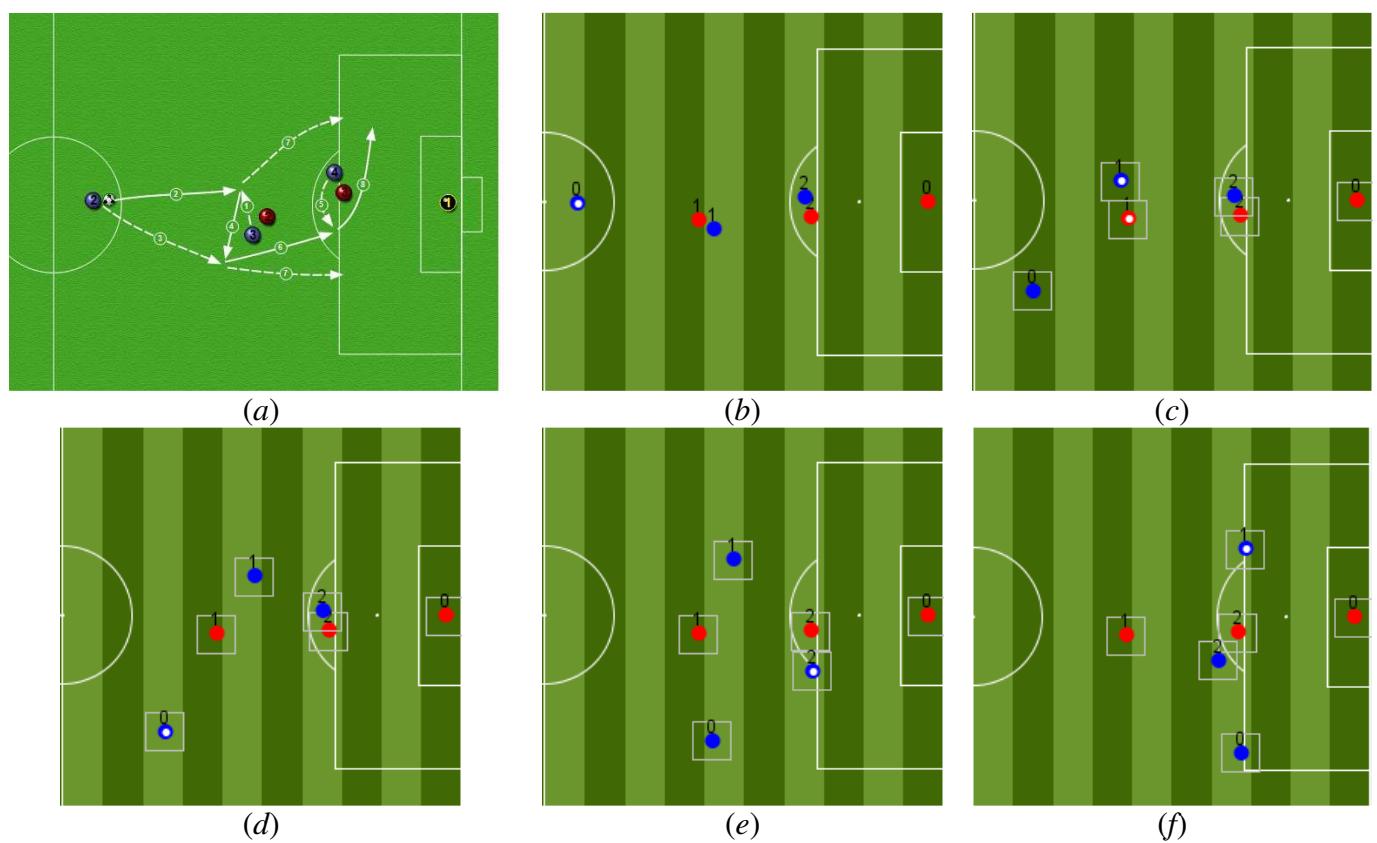
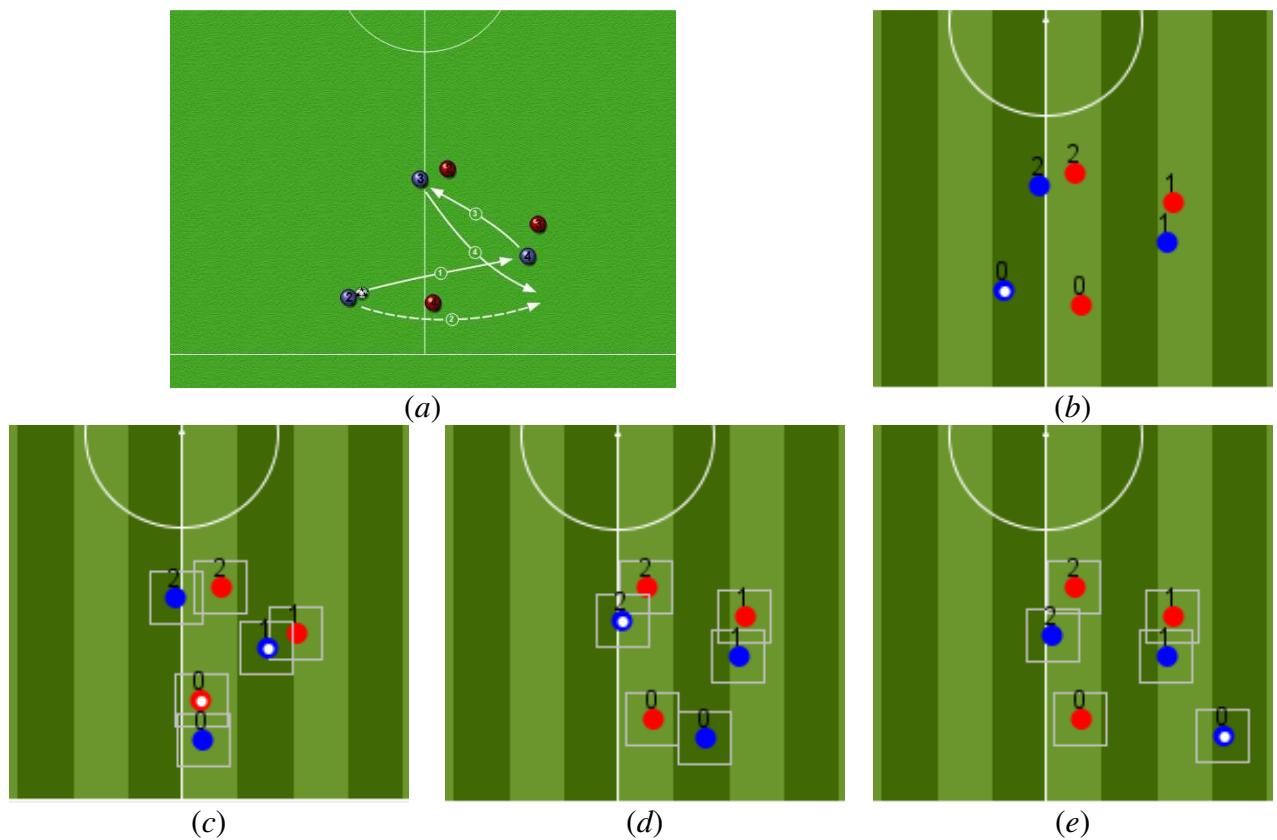


Figure 7.20 – Diagramme et séquence de frames : « negative_space »

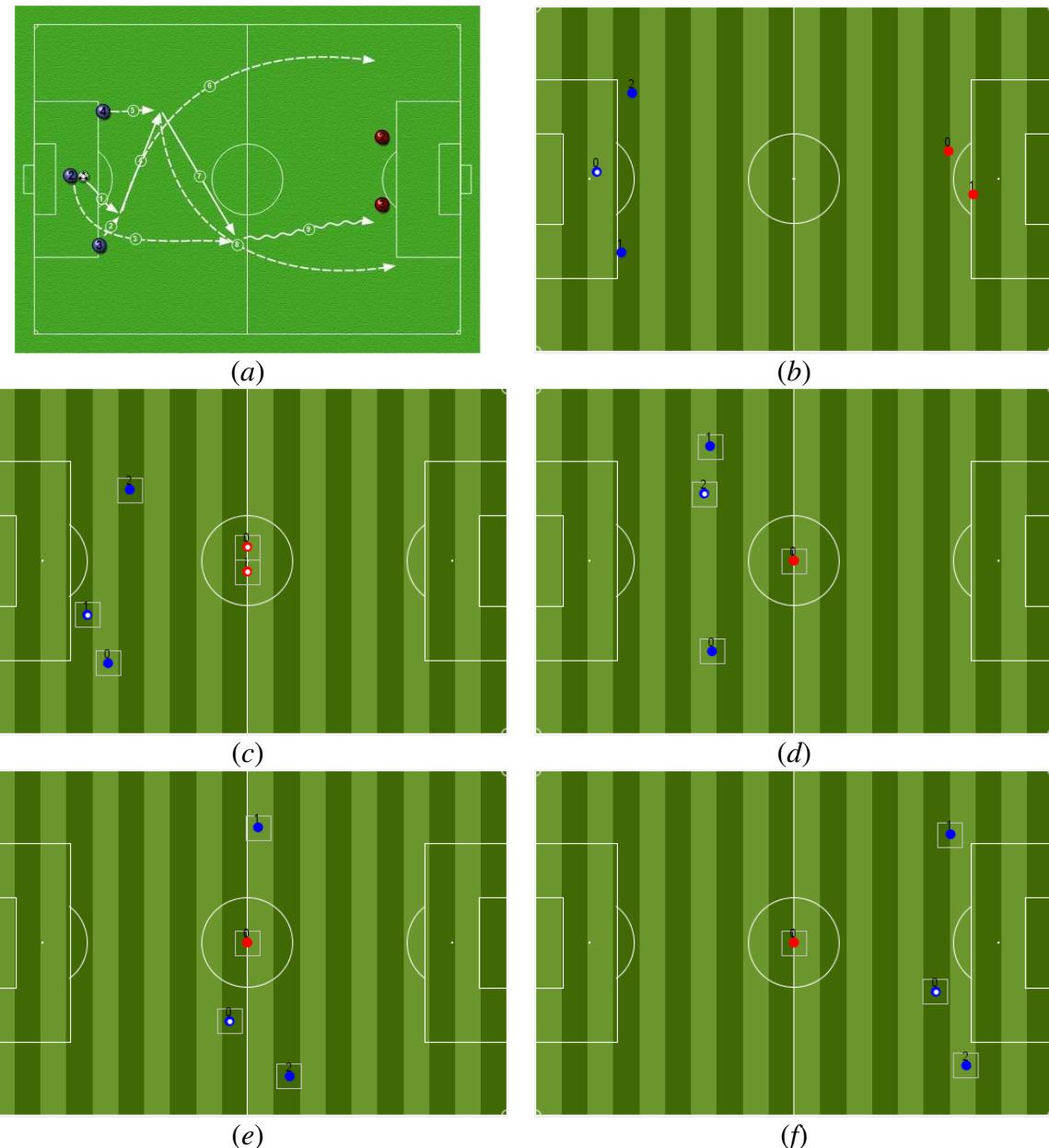
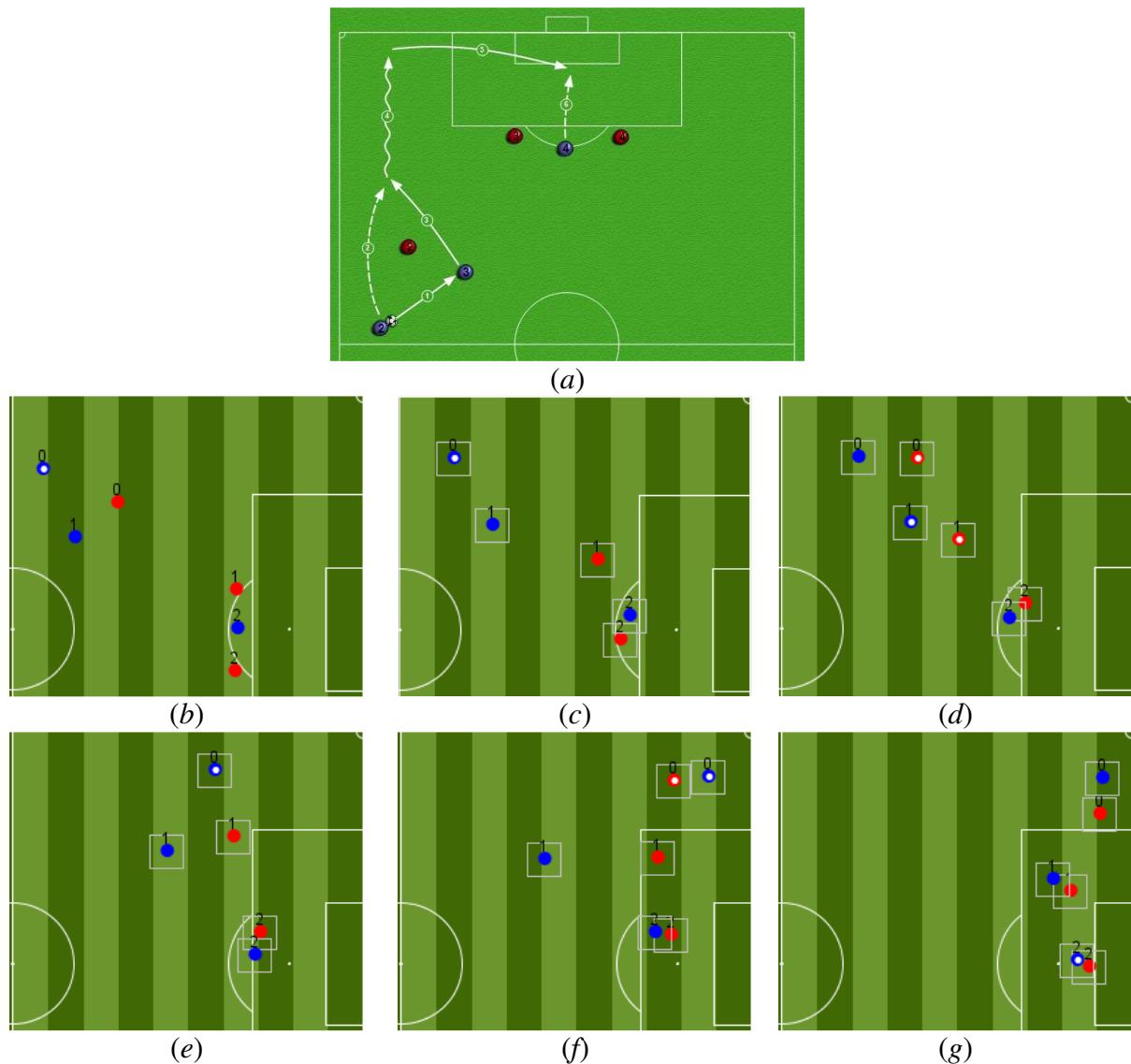


Figure 7.21 – Diagramme et séquence de frames : « *run_pass* »



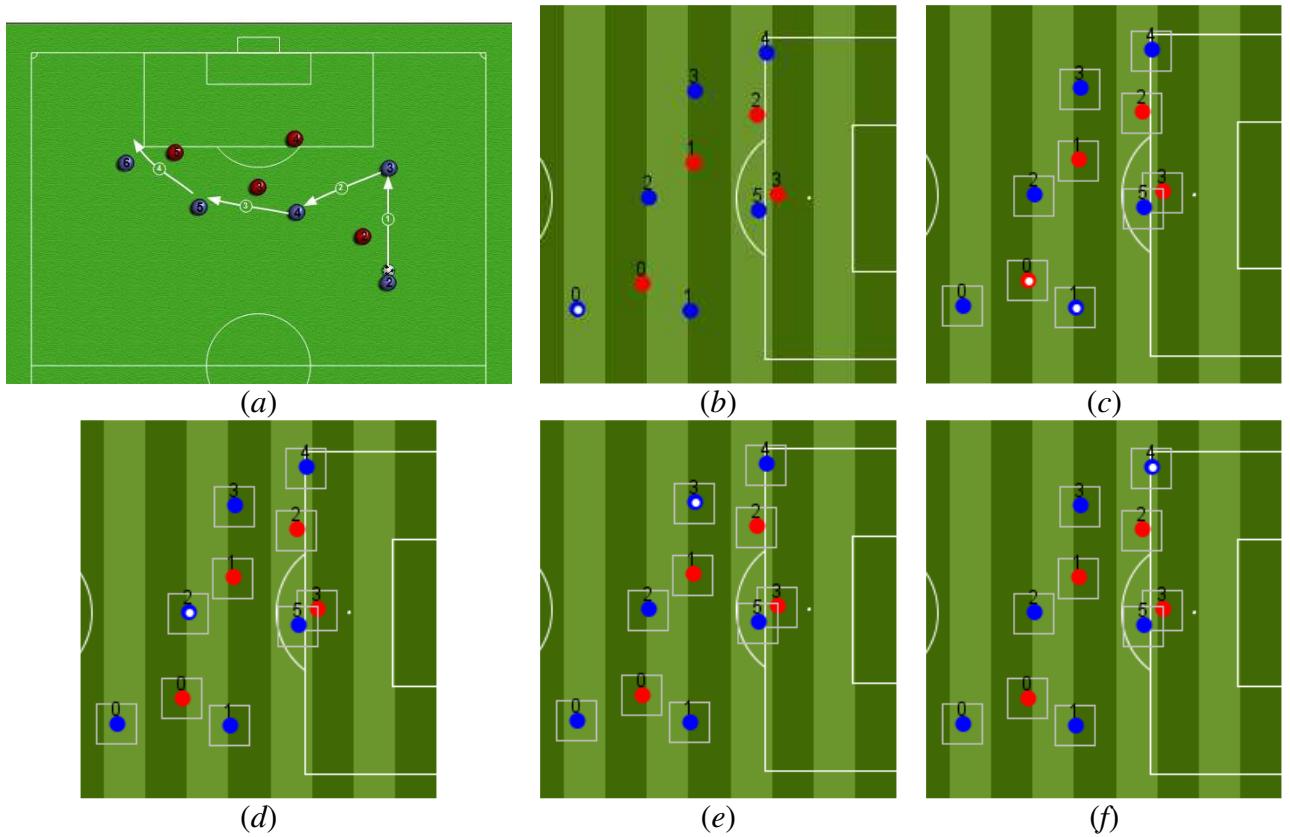


Figure 7.23 – Diagramme et séquence de frames : « turn_attack »

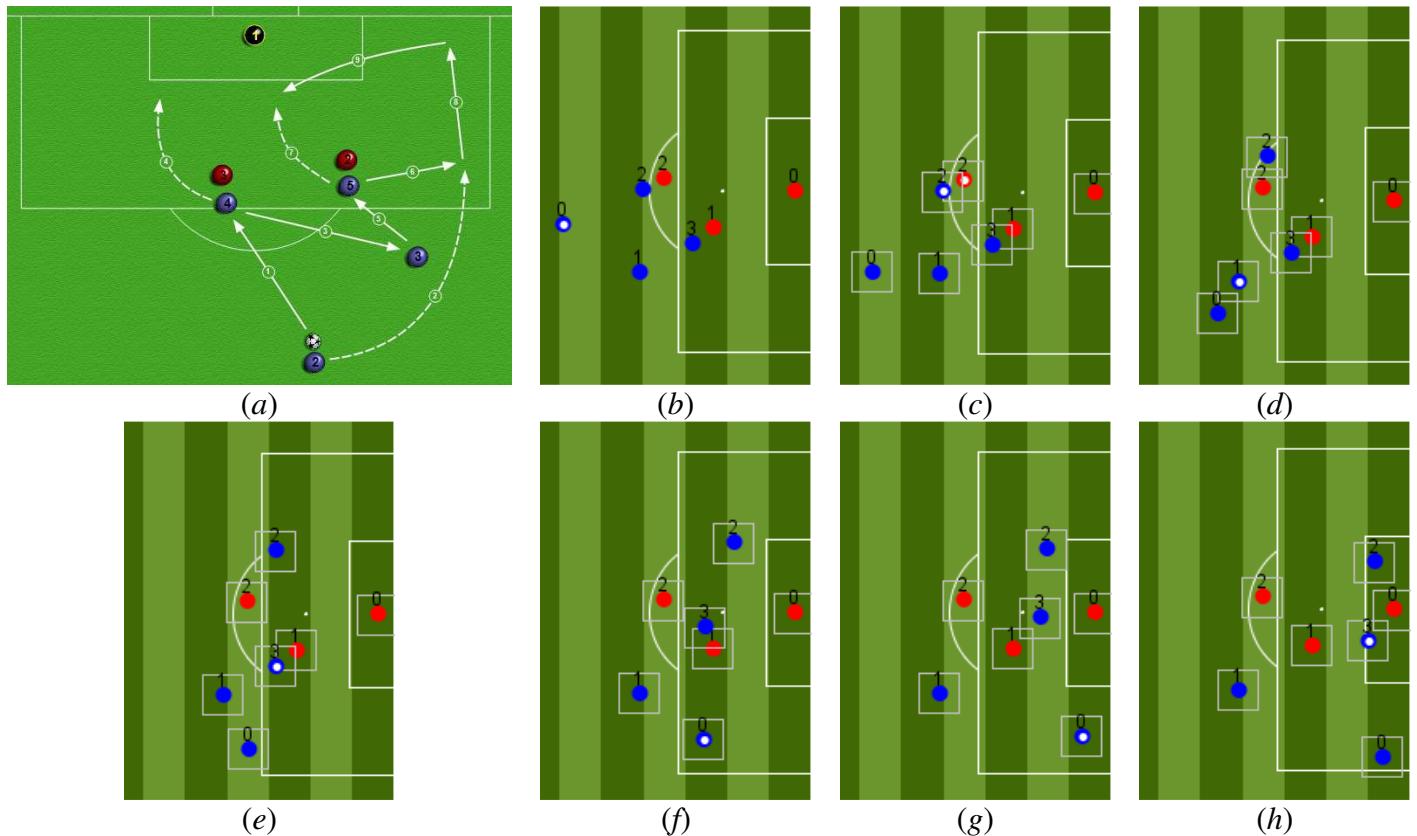


Figure 7.24 – Diagramme et séquence de frames : « turn_cross »

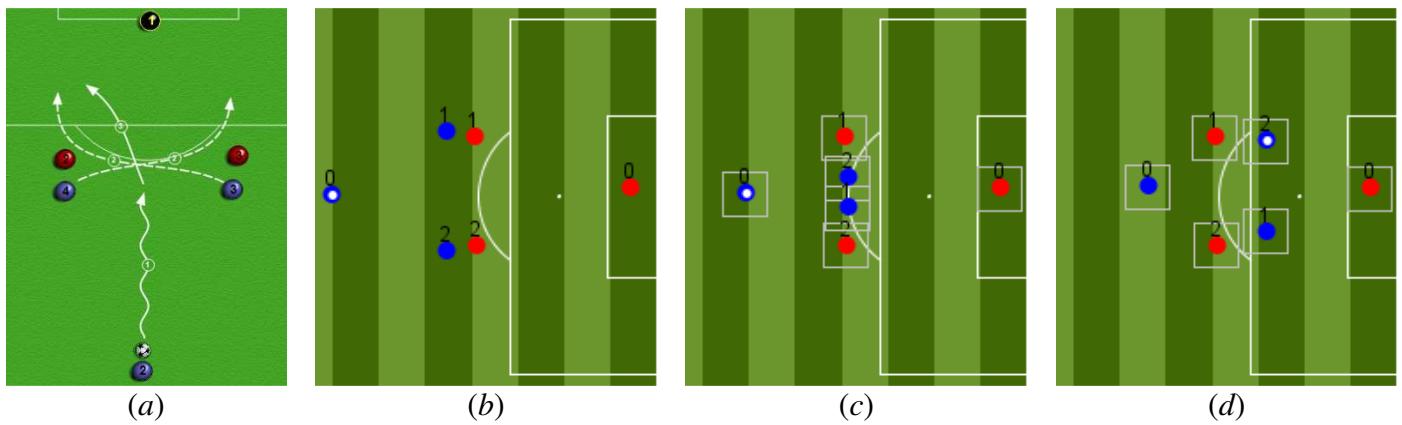


Figure 7.25 – Diagramme et séquence de frames : « *x* »

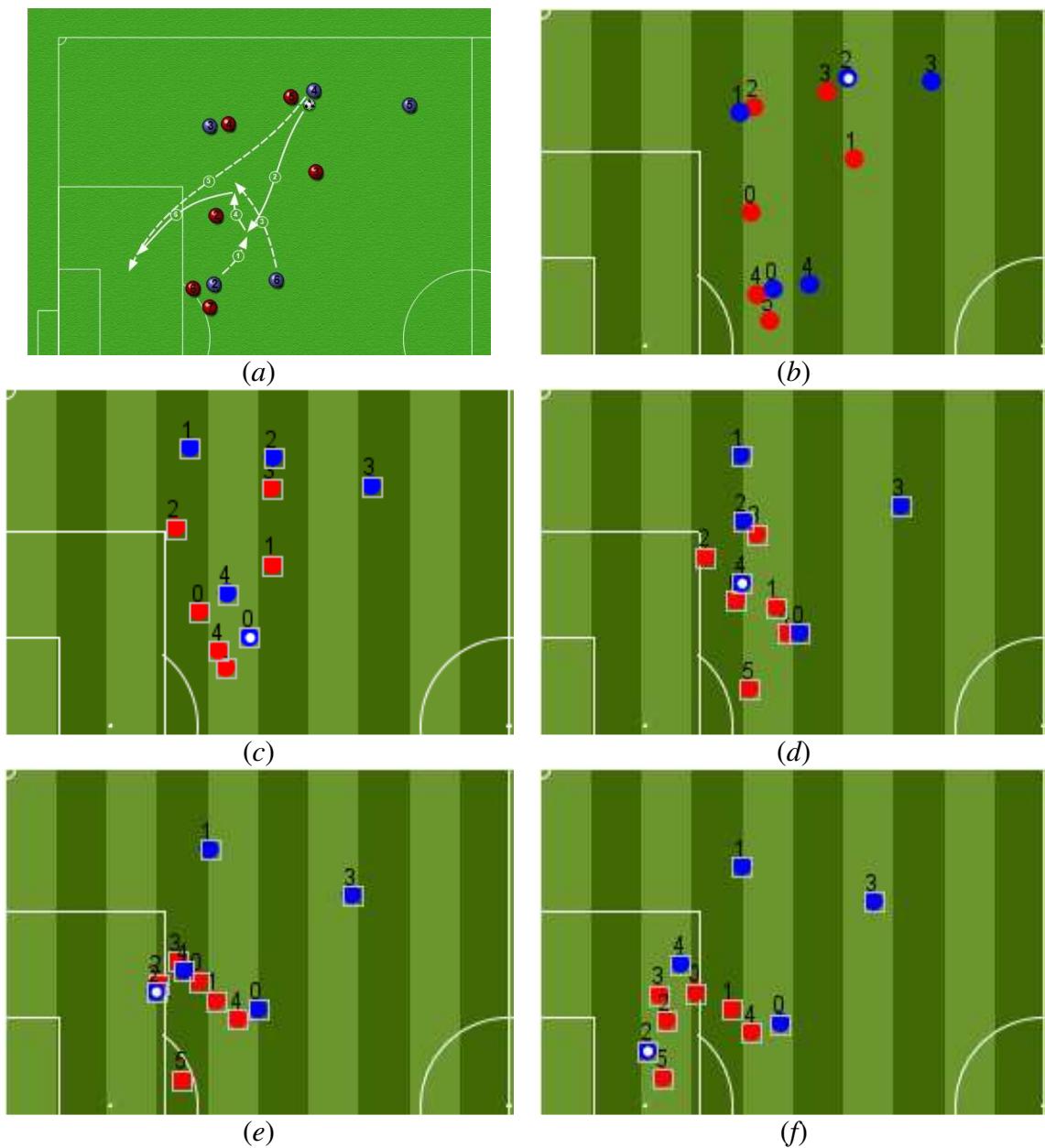


Figure 7.26 – Diagramme et séquence de frames : « *ex_match* »

Annexe E

Structure de données du système.

Les données manipulées par notre système sont sous le format XML, nous avons donc créé une représentation pour la description d'une situation de jeu et une représentation pour la numérisation d'un match. Les XML Schema et deux exemples très simples sont montrés dans les sections suivantes.

E.1 XML Scheme d'une situation de jeu

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="intention" type="IntentionType"/>
    <xsd:complexType name="IntentionType">
        <xsd:sequence>
            <xsd:element name="description" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="comment" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="field" type="Field"/>
            <xsd:element name="team" type="Team" minOccurs="1"
maxOccurs="unbounded"/>
            <xsd:element name="objectives" type="Objectives"/>
        </xsd:sequence>
    </xsd:complexType>
```

```

<xsd:complexType name="Field">
    <xsd:sequence>
        <xsd:element name="condition" type="xsd:string"/>
        <xsd:element name="width" type="xsd:unsignedByte"/>
        <xsd:element name="length" type="xsd:unsignedByte"/>
        <xsd:element name="type" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Team">
    <xsd:attribute name="teamNum" type="xsd:unsignedByte"/>
    <xsd:attribute name="player" type="xsd:unsignedByte"/>
</xsd:complexType>

<xsd:complexType name="Objectives">
    <xsd:sequence>
        <xsd:element name="frame" type="Frame" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Frame">
    <xsd:sequence>
        <xsd:element name="objective" type="Objective"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="frameNum" type="xsd:unsignedInt"/>
</xsd:complexType>

<xsd:complexType name="Objective">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:float"/>
        <xsd:element name="y" type="xsd:float"/>
        <xsd:element name="z" type="xsd:float"/>
        <xsd:element name="width" type="xsd:float"/>
        <xsd:element name="length" type="xsd:float"/>
        <xsd:element name="withBall" type="xsd:boolean"/>
        <xsd:element name="isConsidered" type="xsd:boolean"/>
    </xsd:sequence>
    <xsd:attribute name="playerNum" type="xsd:unsignedByte"/>
    <xsd:attribute name="teamNum" type="xsd:unsignedByte"/>
</xsd:complexType>

</xsd:schema>

```

E.2 Exemple d'une situation de jeu

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Soccer description file --&gt;
&lt;intention&gt;
    &lt;!--
        - Match general description
        - date, start, end
    --&gt;
</pre>

```

```

<description>Description of match, player A gets the ball and
kick</description>
<comment>Anything about this match...</comment>

<!--
*      Playing field description
*          condition (seco, molhado, gelo)
*          size
*              width 45-90 m
*              length 90-120 m
*              for international matches
*              width 64-75 m
*              length 100-110 m
*          type (grass,synthetic,sand)
-->
<field>
    <condition>ice</condition>
    <width>90</width>
    <length>120</length>
    <type>grass</type>
</field>

<!--
*      Team description
*          id
*          number      - list of players
-->
<team teamNum = "1" players="5"/>
<team teamNum = "2" players="4"/>

<!--
*      List of objectives
-->
<frames>
    <frame frameNum="1">
        <objective playerNum="1" teamNum="1">
            <x>-5</x> <y>0</y> <z>0</z>
            <width>5</width>
            <length>5</length>
            <withBall>true</withBall>
        </objective>
        <objective playerNum="2" teamNum="1">
            <x>-10</x> <y>10</y> <z>0</z>
            <width>5</width>
            <length>5</length>
        </objective>
        <objective playerNum="3" teamNum="1">
            <x>-10</x> <y>20</y> <z>0</z>
            <width>5</width>
            <length>5</length>
        </objective>
        <objective playerNum="1" teamNum="2">
            <x>0</x> <y>0</y> <z>0</z>
            <width>5</width>
            <length>5</length>
        </objective>
        <objective playerNum="2" teamNum="2">
            <x>10</x> <y>10</y> <z>0</z>
            <width>5</width>
            <length>5</length>
        </objective>
    </frame>
</frames>

```

```

<objective playerNum="3" teamNum="2">
  <x>10</x> <y>20</y> <z>0</z>
  <width>5</width>
  <length>5</length>
</objective>
</frame>
<frame frameNum="2">
  <objective playerNum="1" teamNum="1">
    <x>-5</x> <y>0</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="2" teamNum="1">
    <x>-10</x> <y>10</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="3" teamNum="1">
    <x>-10</x> <y>20</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="1" teamNum="2">
    <x>0</x> <y>0</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="2" teamNum="2">
    <x>10</x> <y>10</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="3" teamNum="2">
    <x>10</x> <y>20</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
</frame>
<frame frameNum="3">
  <objective playerNum="1" teamNum="1">
    <x>-5</x> <y>0</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="2" teamNum="1">
    <x>-10</x> <y>10</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="3" teamNum="1">
    <x>-10</x> <y>20</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="1" teamNum="2">
    <x>0</x> <y>0</y> <z>0</z>
    <width>5</width>
    <length>5</length>
  </objective>
  <objective playerNum="2" teamNum="2">
    <x>10</x> <y>10</y> <z>0</z>
  </objective>
</frame>

```

```

        <width>5</width>
        <length>5</length>
    </objective>
    <objective playerNum="3" teamNum="2">
        <x>10</x> <y>20</y> <z>0</z>
        <width>5</width>
        <length>5</length>
    </objective>
</frame>
</frames>
</intention>

```

E.3 XML Scheme d'une partie d'un match

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="match" type="MatchType"/>
    <xsd:complexType name="MatchType">
        <xsd:sequence>
            <xsd:element name="description" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="comment" type="xsd:string"
minOccurs="0"/>
            <xsd:element name="date" type="xsd:dateTime"
minOccurs="0"/>
            <xsd:element name="duration" type="xsd:time"
minOccurs="0"/>
            <xsd:element name="field" type="Field"/>
            <xsd:element name="ball" type="Ball"/>
            <xsd:element name="referees" type="Referees"
minOccurs="0"/>
            <xsd:element name="team" type="Team" minOccurs="1"
maxOccurs="unbounded"/>
                <xsd:element name="frames" type="Frames"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="Field">
            <xsd:sequence>
                <xsd:element name="condition" type="xsd:string"/>
                <xsd:element name="width" type="xsd:unsignedByte"/>
                <xsd:element name="length" type="xsd:unsignedByte"/>
                <xsd:element name="type" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="Ball">
            <xsd:sequence>
                <xsd:element name="mass" type="xsd:unsignedByte"/>
                <xsd:element name="circumference"
type="xsd:unsignedByte"/>
                <xsd:element name="pressure" type="xsd:unsignedByte"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="Referees">

```

```

<xsd:sequence>
    <xsd:element name="referee" type="xsd:string"/>
    <xsd:element name="assistant1" type="xsd:string"/>
    <xsd:element name="assistant2" type="xsd:string"/>
    <xsd:element name="fourth_official" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Team">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="coach" type="xsd:string"/>
        <xsd:element name="player" type="Player"
maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="teamNum" type="xsd:unsignedByte"/>
</xsd:complexType>

<xsd:complexType name="Frames">
    <xsd:sequence>
        <xsd:element name="frame" type="Frame" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Frame">
    <xsd:sequence>
        <xsd:element name="time" type="xsd:time" minOccurs="0"/>
        <xsd:element name="playerPosition" type="PlayerPosition"
maxOccurs="unbounded"/>
        <xsd:element name="ballPosition" type="BallPosition"/>
    </xsd:sequence>
    <xsd:attribute name="frameNum" type="xsd:unsignedInt"/>
</xsd:complexType>

<xsd:complexType name="Player">
    <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="mass" type="xsd:unsignedByte"/>
        <xsd:element name="height" type="xsd:unsignedByte"/>
        <xsd:element name="shirt" type="xsd:unsignedByte"/>
    </xsd:sequence>
    <xsd:attribute name="playerNum" type="xsd:unsignedByte"
use="required"/>
    <xsd:attribute name="goalkeeper" type="xsd:boolean"
use="optional" default="false"/>
</xsd:complexType>

<xsd:complexType name="PlayerPosition">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:float"/>
        <xsd:element name="y" type="xsd:float"/>
        <xsd:element name="z" type="xsd:float"/>
    </xsd:sequence>
    <xsd:attribute name="playerNum" type="xsd:unsignedByte"/>
    <xsd:attribute name="teamNum" type="xsd:unsignedByte"/>
</xsd:complexType>

<xsd:complexType name="BallPosition">
    <xsd:sequence>
        <xsd:element name="x" type="xsd:float"/>

```

```

<xsd:element name="y" type="xsd:float"/>
<xsd:element name="z" type="xsd:float"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>

```

E.4 Exemple d'une partie d'un patch

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Soccer description file created by XMLConverter --&gt;
&lt;match&gt;
    &lt;!--
        -      Match general description
        -          date, start, end
    --&gt;
    &lt;description&gt;"File created by XMLConverter"&lt;/description&gt;
    &lt;comment&gt;"IMPORTANT: Default values for fields not
matching..."&lt;/comment&gt;
    &lt;date&gt;"1999-10-20-05:00"&lt;/date&gt;
    &lt;duration&gt;"90:00:00"&lt;/duration&gt;

    &lt;!--
        *      Playing field description
        *          condition (dry, wetted, ice)
        *          size
        *              width 45-90 m
        *              length 90-120 m
        *              for <u>international matches
        *              width 64-75 m
        *              length 100-110 m
        *          type (grass, synthetic, sand)
    -->
    <field>
        <condition>dry</condition>
        <width>90</width>
        <length>120</length>
        <type>grass</type>
    </field>

    <!--
        *      Ball description
        *          mass 410-450 g
        *          circumference 68-70 cm
        *          pressure 0.6-1.1 atm
    -->
    <ball>
        <mass>430</mass>
        <circumference>69</circumference>
        <pressure>1</pressure>
    </ball>

    <!--
        *      Referees description
        *          Name of referee, assitants and fourt oficial
    -->
    <referees>

```

```

<referee>PReferee</referee>
<assistant1>Assistant 1</assistant1>
<assistant2>Assistant 2</assistant2>
<fourth_official>Fourth Official</fourth_official>
</referees>

<!--
*      Team description
*          id
*          name
*          coach
*          player - list of players
*              id
*              name
*              length - length of player in meters
*              mass - mass of player in grames
*              shirt - shirt number
*              goalkeeper - the goalkeeper flag
-->
<team teamNum = "1">
    <name>Team 1</name>
    <coach>Coach</coach>
    <player playerNum = "1">
        <name>Player 1</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>1</shirt>
    </player>
    <player playerNum = "2">
        <name>Player 2</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>2</shirt>
    </player>
    <player playerNum = "3">
        <name>Player 3</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>3</shirt>
    </player>
    <player playerNum = "4">
        <name>Player 4</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>4</shirt>
    </player>
    <player playerNum = "5">
        <name>Player 5</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>5</shirt>
    </player>
</team>
<!--
*      Team description
*          id
*          name
*          coach
*          player - list of players
*              id
*              name
*              length
*              mass
*              shirt
*              goalkeeper
-->
```

```

*
*           length - length of player in meters
*           mass - mass of player in grames
*           shirt - shirt number
*           goalkeeper - the goalkeeper flag
-->
-->
<team teamNum = "2">
    <name>Team 2</name>
    <coach>Coach</coach>
    <player playerNum = "1">
        <name>Player 1</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>1</shirt>
    </player>
    <player playerNum = "2">
        <name>Player 2</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>2</shirt>
    </player>
    <player playerNum = "3">
        <name>Player 3</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>3</shirt>
    </player>
    <player playerNum = "4">
        <name>Player 4</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>4</shirt>
    </player>
    <player playerNum = "5">
        <name>Player 5</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>5</shirt>
    </player>
    <player playerNum = "6">
        <name>Player 6</name>
        <mass>80</mass>
        <height>180</height>
        <shirt>6</shirt>
    </player>
</team>

<!--
*      List of match frames
-->
<frames>
    <frame frameNum="0">
        <time>"00:00:00"</time>
        <playerPosition playerNum="1" teamNum="1">
            <x>-27.695</x> <y>-4.695</y> <z>0.0</z>
        </playerPosition>
        <playerPosition playerNum="2" teamNum="1">
            <x>-30.042</x> <y>-23.507</y> <z>0.0</z>
        </playerPosition>
        <playerPosition playerNum="3" teamNum="1">
            <x>-21.539</x> <y>-28.718</y> <z>0.0</z>
    </frame>
</frames>
```

```

        </playerPosition>
<playerPosition playerNum="4" teamNum="1">
    <x>-11.122</x> <y>-27.541</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="5" teamNum="1">
    <x>-24.387</x> <y>-6.343</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="1" teamNum="2">
    <x>-30.808</x> <y>-14.518</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="2" teamNum="2">
    <x>-19.002</x> <y>-19.597</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="3" teamNum="2">
    <x>-28.873</x> <y>-24.84</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="4" teamNum="2">
    <x>-22.995</x> <y>-26.772</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="5" teamNum="2">
    <x>-29.169</x> <y>-5.016</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="6" teamNum="2">
    <x>-28.56</x> <y>-2.6</y> <z>0.0</z>
</playerPosition>
<ballPosition>
    <x>-20.745</x> <y>-27.905</y> <z>0.0</z>
</ballPosition>
</frame>
<frame frameNum="1">
<time>"00:00:00"</time>
<playerPosition playerNum="1" teamNum="1">
    <x>-27.464</x> <y>-5.186</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="2" teamNum="1">
    <x>-30.328</x> <y>-23.79</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="3" teamNum="1">
    <x>-21.279</x> <y>-28.673</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="4" teamNum="1">
    <x>-11.382</x> <y>-27.449</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="5" teamNum="1">
    <x>-24.387</x> <y>-6.343</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="1" teamNum="2">
    <x>-30.808</x> <y>-14.518</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="2" teamNum="2">
    <x>-19.066</x> <y>-19.661</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="3" teamNum="2">
    <x>-29.012</x> <y>-24.979</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="4" teamNum="2">
    <x>-22.741</x> <y>-26.515</y> <z>0.0</z>
</playerPosition>
<playerPosition playerNum="5" teamNum="2">
    <x>-29.129</x> <y>-5.34</y> <z>0.0</z>
</playerPosition>

```

```
<playerPosition playerNum="6" teamNum="2">
  <x>-28.56</x> <y>-2.6</y> <z>0.0</z>
</playerPosition>
<ballPosition>
  <x>-20.563</x> <y>-27.718</y> <z>0.0</z>
</ballPosition>
</frame>
</frames>
</match>
```


Annexe F

Sensibilité du transfert des valeurs à la qualité de la politique de transfert.

La Figure 7.27 montre bien comme les méthodes de transfert de connaissances basées sur le transfert de valeurs sont très sensibles à la qualité de la connaissance transférée. Sur l'axe des abscisses nous avons le nombre d'épisodes d'apprentissage et sur l'axe des ordonnées nous avons la moyenne du gain obtenu sur 30 expérimentations. La figure présente quatre courbes : la courbe représentée par la ligne pleine montre le résultat obtenu par l'algorithme standard *Q-Learning* sans l'utilisation du transfert de connaissance ; les trois autres utilisent le transfert de valeurs comme méthode de transfert de connaissance. Ces trois derniers partent d'une politique 40-altérée pour un espace d'état de 400, cela veut dire que les politiques sont 10% altérées ou encore que 90% de la politique utilisée pour le transfert est en accord avec la politique optimale. Cependant ces politiques sont altérées d'une manière différente par rapport aux changements de la valeur des états perturbés.

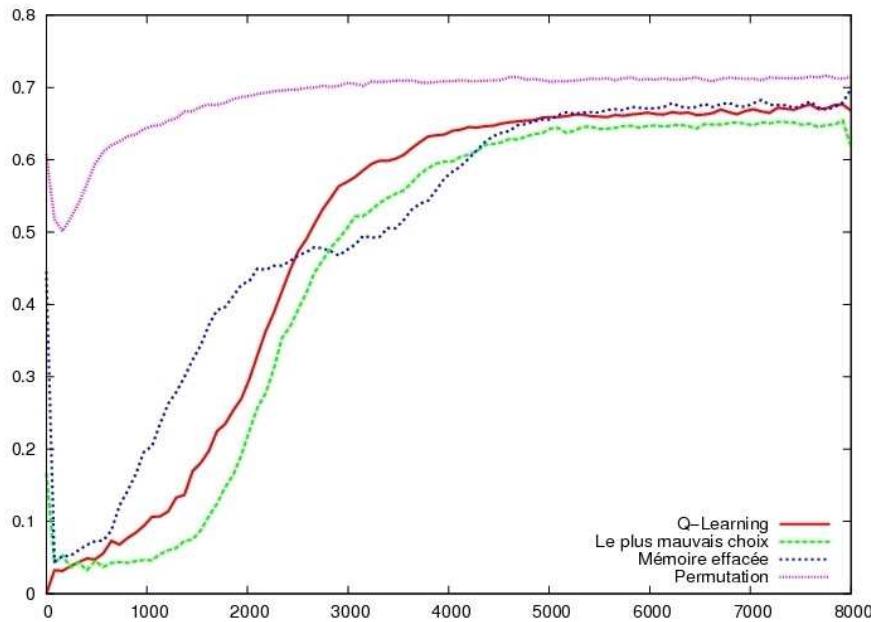


Figure 7.27 – Sensibilité du transfert des valeurs à la qualité de la source

Autrement dit, les trois politiques altérées ont exactement les mêmes états perturbés, mais ces états sont perturbés différemment selon les procédés suivants :

- Le plus mauvais choix : dans ce cas la valeur donnée pour le meilleur choix (action) est attribuée au plus mauvais choix et les autres valeurs sont effacées. Par exemple, supposons que la meilleure action est d'aller vers le nord, avec cette perturbation la meilleure action sera d'aller vers le sud.
- Mémoire effacée : cette perturbation remet la valeur des états perturbés à leur valeur de départ. On refait la procédure d'initialisation des états pour les états perturbés.
- Permutation : la valeur perturbée est choisie aléatoirement parmi les valeurs actuelles de la politique optimale. Cela veut dire que la nouvelle valeur (état perturbé) est choisie aléatoirement dans l'intervalle de valeurs valides de la politique optimale.

Avec ce que nous venons de présenter et les courbes de la Figure 7.27 nous pouvons voir que les méthodes de transfert des valeurs sont très sensibles à la politique transférée. Nous avons utilisé trois politiques avec les mêmes 10% d'états perturbés, ou une politique qui est 90% en conformité avec la politique optimale, et nous avons obtenu des résultats qui vont d'un apprentissage bien plus rapide (la courbe « Permutation » est beaucoup meilleur que le Q-Learning) à la catastrophe d'être plus lent que l'algorithme standard du Q-Learning (courbe « Le plus mauvais choix »).

