# Apprentissage par renforcement factorisé pour le comportement de personnages non joueurs

**Thomas Degris*** — **Olivier Sigaud**** — **Pierre-Henri Wuillemin*****

* *3-34 Athabasca Hall — Department of Computing Science — University of Alberta — Edmonton, AB, Canada, T6G 2E8*

*degris@cs.ualberta.ca*

** *Institut des Systèmes Intelligents et de Robotique — Université Pierre et Marie Curie - Paris 6, CNRS UMR 7222 — Pyramide - Tour 55, Boîte courrier 173 — 4 place Jussieu, F75252 Paris Cedex 05*

*olivier.sigaud@upmc.fr*

*** *Laboratoire d'Informatique de Paris 6 — Université Pierre et Marie Curie - Paris 6, CNRS UMR 7606 — 4 place Jussieu, F75252 Paris Cedex 05*

*pierre-henri.wuillemin@lip6.fr*

RÉSUMÉ. *Dans cet article, nous appliquons une méthode générale d'apprentissage par renforcement pour la mise au point automatique de comportements de personnages non joueurs d'un jeu vidéo de tir à la première personne, Counter-Strike©. Le résultat de l'apprentissage est un ensemble d'arbres de décision représentant de façon lisible un modèle du problème et la politique de décision des personnages. Enfin, nous discutons de la portée de notre méthode pour la réalisation d'architectures de décision pour les personnages non joueurs de jeux vidéo.*

ABSTRACT. *In this paper, we apply a general reinforcement learning method to automatically design the behavior of non player characters of the Counter-Strike© first person shooter computer game. The result of the learning process is a set of decision trees that represents compactly and easily readable a model of the problem itself and the decision policy of characters. Beyond this example, we discuss the potential benefits of our method to design the decision architecture of non player characters in commercial computer games.*

MOTS-CLÉS : *Apprentissage par renforcement, Factorisation, Personnages non joueurs.*

KEYWORDS: *Reinforcement learning, Factorization, Non player characters.*

RSTI - RIA - 23/2009. Jeux : modélisation et décision, pages 221 à 244

## 1. Introduction

Les jeux vidéo tels que les jeux de tir en première personne[1] constituent pour l'intelligence artificielle (IA) un domaine d'application à la fois très exigeant et très complet (Laird *et al.*, 2000). En effet, les agents logiciels immergés dans de tels jeux, appelés personnages non joueurs (PNJ), doivent résoudre de nombreux problèmes (navigation, réalisation de séquences d'actions appropriées, coordination lorsqu'ils sont en équipe, etc.) sous des contraintes fortes de réactivité, tout en disposant d'un temps de calcul très limité.

Par ailleurs, de tels jeux proposent pour les méthodes adaptatives développées en IA un environnement d'évaluation particulièrement favorable. On y trouve en effet une richesse dans les perceptions et les possibilités d'action comparable à celle des robots, mais on peut répéter les expériences un grand nombre de fois sans pour autant se heurter aux difficultés spécifiques de la robotique, telles que la nécessité de se localiser, le bruit sur les capteurs et les actionneurs, les pannes diverses, le coût du matériel ou encore les problèmes d'autonomie énergétique (Sigaud, 2004).

Dans ce contexte, diverses méthodes d'apprentissage ont été appliquées au cadre des jeux vidéo, la plupart relevant de l'apprentissage par renforcement (voir (Sutton *et al.*, 1998) pour une présentation de l'apprentissage par renforcement et (Madeira, 2007) pour une présentation générale de l'apprentissage dans les jeux vidéo). Une approche élémentaire appelée *Dynamic Scripting* (Spronck *et al.*, 2006) consiste simplement à adapter les poids d'un ensemble de règles en fonction du succès du comportement correspondant. Cette approche a été appliquée notamment à des règles décrivant des comportement de combat dans NeverWinterNight©. Une approche plus élaborée consiste à utiliser des systèmes de classeurs (Sigaud, 2007) qui découvrent automatiquement de nouvelles règles en plus d'adapter leurs poids. Dans ce cadre, on peut citer en particulier les travaux de Robert (Robert, 2005) qui définit une architecture motivationnelle à base de systèmes de classeurs appliquée avec succès à l'apprentissage du comportement de PNJ pour le jeu Team Fortress Classic©.

Depuis quelques années, pourtant, des avancées ont été enregistrées en apprentissage par renforcement avec les processus décisionnels de Markov factorisés (FMDP) (Boutilier *et al.*, 1995; Boutilier *et al.*, 2000). Ce formalisme permet de définir un problème en précisant les dépendances entre les variables décrivant l'état de celui-ci. Ces dépendances sont ensuite exploitées pour faciliter la recherche d'une solution. Mais ces techniques restent limitées par la nécessité de fournir manuellement une description formelle de la tâche à accomplir, opération difficilement réalisable dans le cadre d'un problème réellement complexe.

Dans un cadre théorique proche des FMDP, (Guestrin *et al.*, 2003) utilise les MDP relationnels, *Relational* MDP (RMDP), pour résoudre un problème de planification dans

1. en anglais *First Person Shooters (FPS)*

le jeu de stratégie temps réel Stratagus[2]. Cependant, ces travaux supposent aussi une connaissance complète a priori de la structure du problème.

Très récemment, nous avons mis au point une méthode qui permet d'apprendre la description formelle d'une tâche sous la forme d'un FMDP plutôt que de la fournir manuellement (Degris, 2007). Notre approche est fonctionnellement très proche des systèmes de classeurs, mais elle bénéficie de tout l'arrière plan théorique du cadre des FMDP là où les systèmes de classeurs utilisent des heuristiques fondées sur les algorithmes génétiques (Sigaud *et al.*, 2009). Nous disposons donc d'une méthode générique d'apprentissage par renforcement pour les problèmes de grande taille dont nous allons montrer au travers d'un exemple simple qu'elle peut être appliquée au problème de la mise au point de comportements pour les PNJ du jeu vidéo Counter-Strike©.

Notre contribution est organisée de la façon suivante. Dans la section 2, nous présentons le cadre théorique dans lequel s'inscrivent nos travaux puis, dans la section 3, nous présentons les outils que nous avons développés. Nous décrivons ensuite à la section 4 le jeu Counter-Strike© qui nous sert de cadre applicatif et la représentation des problèmes d'apprentissage que nous avons abordés. Nous présentons section 5 nos résultats expérimentaux. Enfin, nous discutons à la section 6 la portée de ces résultats pour la mise au point d'une IA pour des jeux vidéo du commerce, avant de récapituler les principaux enseignements de notre travail en conclusion.

## 2. Cadre théorique

Les Processus Décisionnels de Markov (MDP) constituent un cadre privilégié pour modéliser des problèmes de planification et d'apprentissage par renforcement dans l'incertain (Puterman, 1996). On parle de planification lorsque toutes les données qui décrivent un MDP sont connues a priori et d'apprentissage si ce n'est pas le cas. Dans cette section, nous décrivons brièvement le cadre des MDP et son extension factorisée nommée FMDP, puis les méthodes d'apprentissage par renforcement indirect.

### 2.1. *Processus Décisionnels de Markov*

Un MDP est un tuple $\langle S, A, R, T \rangle$, où $S$ et $A$ sont des ensembles finis d'états et d'actions d'un agent ; $R : S \times A \rightarrow \mathbb{R}$ est la fonction de récompense immédiate $R(s, a)$ et $T : S \times A \rightarrow \Pi(S)$ est la fonction décrivant les probabilités de transition $P(s'|s, a)$ du MDP. Réaliser l'action $a \in A$ dans l'état $s \in S$ produit la transition du système dans l'état $s' \in S$ avec une probabilité $P(s'|s, a)$ et donne lieu à la récompense $R(s, a)$.

Une politique stationnaire $\pi : S \times A \rightarrow [0, 1]$ définit la probabilité $\pi(s, a)$ que l'agent fasse l'action $a$ dans l'état $s$. Le but de la planification et de l'apprentissage

2. http ://www.stratagus.org/

est de trouver une politique $\pi$ qui maximise la fonction de valeur $V_\pi(s)$ utilisant le critère de récompense actualisée : $V_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t, r_t | s_0 = s]$ où $\gamma \in [0, 1[$ est le facteur d'actualisation, $r_t$ la récompense obtenue à l'instant $t$ et $s_0$ l'état initial, en considérant un horizon infini. La fonction de valeur d'action $Q_\pi(s, a)$ est définie par :

$$Q_\pi(s, a) = \sum_{s' \in S} P(s'|s, a)(R(s', a) + \gamma V_\pi(s')) \quad [1]$$

Une politique $\pi$ est optimale si, pour tout $s \in S$ et toute politique $\pi'$, $V_\pi(s) \geq V_{\pi'}(s)$. La fonction de valeur d'une politique optimale est appelée fonction de valeur optimale et notée $V^*$.

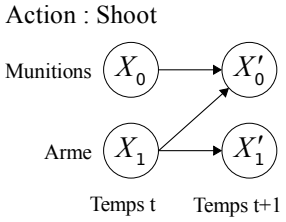
Un problème satisfait l'hypothèse de Markov lorsque l'égalité des probabilités suivantes est vérifiée : $P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t)$. Lorsque la fonction de récompense $R$ et la fonction de transition $T$ sont parfaitement connues et lorsque l'hypothèse de Markov est vérifiée, diverses méthodes itératives relevant de la programmation dynamique peuvent être utilisées pour calculer $V^*$. *Value Iteration* est l'une de ces méthodes, nous renvoyons le lecteur à (Buffet *et al.*, 2008) pour une présentation.

### 2.2. *Processus Décisionnels de Markov factorisés*

Un MDP factorisé (FMDP) (Boutilier *et al.*, 1995) est décrit par un ensemble de variables d'états $S = \{X_1, \ldots, X_n\}$, où chaque $X_i$ prend ses valeurs dans un domaine fini $Dom(X_i)$. Un état $s \in S$ assigne une valeur $x_i \in Dom(X_i)$ à chaque variable d'état $X_i$. Le modèle de transition d'état $T_a$ d'une action $a$ est défini par le graphe de transition $G_a$ representé par un réseau bayésien dynamique (DBN pour *Dynamic Bayes Net*) (Dean *et al.*, 1989). $G_a$ est un graphe orienté acyclique à deux couches dont les nœuds sont $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$ où $X_i$ est une variable au temps $t$ et $X'_i$ la même variable au temps $t + 1$. Nous supposons qu'il n'y a pas d'arcs synchrones, c'est-à-dire pas d'arcs entre des variables au même pas de temps. Un graphe $G_a$ est quantifié par une distribution de probabilités conditionnelle (CPT) associée à chaque nœud $X'_i \in G_a$. Le modèle des transitions $T$ du FMDP est alors défini par un DBN spécifique $T_a = \langle G_a, \{CPT^a_{X_1}, \ldots, CPT^a_{X_n}\}\rangle$ pour chaque action $a$. Représenter le problème sous la forme d'un FMDP permet une réduction potentiellement exponentielle de la taille du modèle.

Par exemple, la figure 1 montre le DBN pour une action *Shoot* et deux variables binaires $X_0$ (indiquant si l'agent a des munitions ou pas) et $X_1$ (indiquant si l'agent a une arme pouvant utiliser les munitions ou pas). Le graphe (figure a) indique les dépendances entre les variables à un pas de temps $t$ et un pas de temps $t+1$. Les figures b et c représentent respectivement CPT $^{Shoot}_{X_0}$ et CPT $^{Shoot}_{X_1}$. On constate par exemple que la CPT de la variable $X_1$ (figure c) est particulièrement compacte puisqu'elle ne dépend que de la valeur de cette même variable au pas de temps précédent.

Des méthodes de planification dédiées aux FMDP ont été développées pour traiter les cas où les fonctions de transition et de récompense sont connues a priori. Nous ne




(a)

| $X_0$ | $X_1$ | $P(X'_0 \mid X_1, X'_0)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0.8 |

(b)

| $X_1$ | $P(X'_1 \mid X'_1)$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

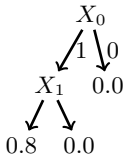
(c)

**Figure 1.** *Exemple de* DBN *pour l'action Shoot et les variables binaires $X_0$ (indiquant si l'agent a des munitions ou pas) et $X_1$ (indiquant si l'agent a une arme pouvant utiliser les munitions). La valeur de $X_0$ au prochain pas de temps dépend des valeurs de $X_0$ et $X_1$ au pas de temps courant. La valeur de $X_1$ au prochain pas de temps ne dépend que de sa valeur au pas de temps courant*

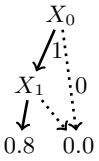
mentionnerons ici que les méthodes de programmation dynamique, à savoir *Structured Policy Iteration* (SPI), *Structured Value Iteration* (SVI) (Boutilier *et al.*, 2000) et *Stochastic Planning Using Decision Diagrams* (SPUDD) (Hoey *et al.*, 1999), qui étendent la programmation dynamique et utilisent respectivement les fonctions du FMDP exprimées sous la forme d'arbres de décision (exemple figure 2.b) ou de diagrammes de décision algébriques (exemple figure 2.c).

| $X_0$ | $X_1$ | $P(X'_0 \mid X_1, X'_0)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0.8 |

(a)




(b)




(c)

**Figure 2.** *Définition de CPT $\frac{Shoot}{X_0}$ (voir figure 1) exprimée sous la forme d'une table (figure a), d'un arbre de décision (figure b) et d'un diagramme de décision algébrique (figure c)*

### 2.3. *Apprentissage par renforcement indirect*

Dans les problèmes où la fonction de transition $T$ et de récompense $R$ sont inconnues, les méthodes d'apprentissage par renforcement indirect proposent d'apprendre un modèle de ces fonctions par des techniques d'apprentissage supervisé et de calculer une politique en appliquant les méthodes de planification décrites précédemment sur ces modèles de plus en plus fidèles. L'architecture DYNA (Sutton, 1990), conçue pour intégrer la planification, l'action et l'apprentissage, est l'archétype de cette approche. L'algorithme DYNA-Q, décrit sur la figure 3, est une instanciation de DYNA qui utilise

l'algorithme *Q-learning* pour approcher $V^*$. Pour cette section, afin de simplifier la description de l'algorithme, nous supposons que le problème est déterministe. Nous utilisons le raccourci d'écriture suivant : $T(s, a)$ représente la fonction renvoyant l'état $s'$ tel que $P(s'|s, a) = 1$.

---

Entrée: $\emptyset$ Sortie: $Q(s, a)$

1) Initialise $Q(s, a)$, $T$ et $R$
2) Jusqu'à convergence :
   a) $s \leftarrow$ état courant (non terminal)
   b) $a \leftarrow \epsilon\text{-}greedy(s, Q)$
   c) Exécute $a$ ; observe $s'$ et $r$
   d) $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
   e) Définit $T(s, a) \rightarrow s'$ et $R(s, a) \rightarrow r$
   f) Répète $N$ fois :
$s \leftarrow$ état aléatoire déjà observé
$a \leftarrow$ action aléatoire déjà testée en $s$
$s' \leftarrow T(s, a)$
$r \leftarrow R(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$

---

**Figure 3.** *L'algorithme* DYNA-Q *(on suppose que le problème est déterministe)*

La phase d'action de DYNA-Q est composée de trois étapes (2.a à 2.c). L'agent suit une politique d'exploration $\epsilon$-*greedy* (Sutton *et al.*, 1998) : à chaque fois qu'une action est décidée (étape 2.b), l'action ayant la valeur d'action la plus élevée est choisie la plupart du temps, avec une probabilité faible de choisir une action de façon aléatoire (l'action est alors sélectionnée suivant une distribution uniforme indépendante des valeurs d'action).

L'étape 2.e représente l'apprentissage supervisé des fonctions $R$ et $T$. L'étape 2.f représente la phase de planification, dans laquelle les modèles de $R$ et $T$ sont exploités pour mettre à jour les valeurs $Q(s, a)$ pour un couple $s \times a$ choisi aléatoirement (dans les couples déjà visités par l'agent, donc existant dans $T$). Il faut noter que DYNA-Q avec $N = 0$ est une approche d'apprentissage par renforcement direct similaire à *Q-learning*. Enfin, DYNA-Q utilise une représentation tabulaire de $Q$, $T$ et $R$, ce qui le rend inopérant sur des problèmes de grande taille, même si la phase de planification est désactivée avec $N = 0$.

## 3. L'architecture SDYNA

Par analogie avec DYNA, (Degris *et al.*, 2006b) ont proposé l'architecture *Structured* DYNA (SDYNA) pour intégrer la planification, l'action et l'apprentissage en utilisant des représentations factorisées plutôt que tabulaires. Cette architecture est conçue

pour résoudre des problèmes d'apprentissage par renforcement de grande taille dont la structure est inconnue a priori, en utilisant des versions incrémentales de techniques de planification fondées sur les FMDP.

L'algorithme 4 décrit SDYNA. $Fact(F)$ désigne une représentation factorisée de la fonction $F$. Différents types de représentations factorisées sont possibles pour exploiter des régularités de $F$, telles que des règles, des arbres de décision, des diagrammes de décision binaires ou algébriques, etc.

Entrée: $Acting, LearnR, LearnT, Plan, Fact$
Sortie: $\{\forall a \in A : Fact(Q_t(s,a))\}$

1) Initialisation
2) A chaque pas de temps $t$, faire :
   a) $s \leftarrow$ état courant (non terminal)
   b) $a \leftarrow Acting(s, \{\forall a \in A : Fact(Q_{t-1}(s,a))\})$
   c) Exécute $a$ ; observe $s'$ et $r$
   d) $Fact(T_t) \leftarrow LearnT(\langle s, a, s' \rangle, Fact(T_{t-1}))$
   e) $Fact(R_t) \leftarrow LearnR(\langle s, a, r \rangle, Fact(R_{t-1}))$
   f) $\{Fact(V_t), \{\forall a \in A : Fact(Q_t(s,a))\}\} \leftarrow$
      $Plan(Fact(R_t), Fact(T_t), Fact(V_{t-1}))$

**Figure 4.** *L'algorithme* SDYNA

La phase d'action de SDYNA (étapes 2.a à 2.c) est similaire à celle de DYNA et des politiques d'exploration telles que $\epsilon$-*greedy* (Sutton *et al.*, 1998) peuvent être utilisées sans modification. Au contraire, les phases d'apprentissage et de planification dépendent directement de la représentation factorisée utilisée. Contrairement à DYNA-Q, SDYNA ne fournit pas de résultat si l'on désactive la phase de planification.

Dans la suite, nous nous focalisons sur une instanciation de SDYNA qui utilise l'induction d'arbres de décision (Quinlan, 1986) pour construire les fonctions manipulées. La représentation arborescente d'une fonction $F$ est notée $Tree(F)$. La phase d'action est instanciée par une politique d'exploration $\epsilon$-*greedy*. Sa phase d'apprentissage (étapes 2.d et 2.e de SDYNA) est fondée sur l'induction incrémentale d'arbres de décision décrite à la section 3.1. Sa phase de planification (étape 2.f) est fondée sur une version incrémentale modifiée de *Structured Value Iteration* (SVI), un algorithme proposé par (Boutilier *et al.*, 2000) et décrit à la section 3.2.

### 3.1. *Apprentissage d'un modèle structuré*

Quand un agent exécute une action $a$ dans un état $s$, il peut exploiter en retour deux informations : son nouvel état $s'$ et sa récompense $r$. Des algorithmes incrémentaux de classification apprennent une fonction à partir d'un ensemble d'exemples $\langle \mathcal{A}, \varsigma \rangle$ où $\mathcal{A}$ est un ensemble d'*attributs* $\nu_i$ et $\varsigma$ est la *classe* de l'exemple. Par conséquent, à partir

de l'observation de l'agent $\langle s, a, s', r\rangle$ avec $s = (x_1, ..., x_n)$ et $s' = (x'_1, ..., x'_n)$, nous proposons d'apprendre un modèle de la fonction de récompense $R(s, a)$ et les modèles de transition $CPT^a_{X_i}$. Concernant l'apprentissage de la fonction de récompense $R(s, a)$, on utilise des exemples $\langle \mathcal{A} = (x_1, ..., x_n, a), \varsigma = r\rangle$. Concernant l'apprentissage de la fonction de transition, nous avons le choix entre deux procédés : soit nous apprenons un arbre de décision pour représenter $CPT^a_{X_i}$ pour chaque variable $X_i$ et chaque action $a$, en présentant à l'algorithme des exemples $\langle \mathcal{A} = (x_1, ..., x_n), \varsigma = x'_i\rangle$ pour chaque action, soit nous apprenons un arbre de décision pour chaque variable $X_i$ en considérant l'action comme une variable de l'état précédent comme les autres, en présentant à l'algorithme des exemples $\langle \mathcal{A} = (x_1, ..., x_n, a), \varsigma = x'_i\rangle$. Les résultats présentés dans cet article ont été obtenus avec la seconde approche.

Nous apprenons donc les modèles des transitions et de la fonction de récompense sous la forme d'arbres de décision. Les exemples successifs reçus par l'agent dans son environnement constituent un flux qui doit être appris de façon incrémentale. Nous avons donc recours à un algorithme incrémental d'induction d'arbres de décision pour l'apprentissage de fonctions stochastiques inspiré de (Schlimmer *et al.*, 1986).

Pour déterminer quel test installer sur chaque nouveau nœud de l'arbre de décision, les algorithmes d'induction incrémentale d'arbres de décision utilisent des mesures tirées de la théorie de l'information, qui sont calculées pour chaque attribut $\nu_i \in \mathcal{A}$. Différentes mesures ont été proposées, telles que *gain ratio*, *Kolmogorov-Smirnoff* ou $\chi^2$ (DasGupta, 2008). Nous utilisons $\chi^2$ car la mesure peut aussi être utilisée en tant que test d'indépendance entre deux distributions de probabilités, comme le suggère (Quinlan, 1986).

Ainsi, nous l'utilisons dans le cadre d'un algorithme de pré-élagage afin d'éviter la construction d'un arbre complet lorsque la fonction à apprendre est stochastique. À une feuille donnée, on calcule le critère $\chi^2$ pour chacune des variables $X_i$ pouvant être sélectionnée. La meilleure mesure calculée est ensuite comparée à une valeur seuil $\tau_{\chi^2}$ pour déterminer si un nouveau nœud de décision est installé. Nous renvoyons le lecteur à (Degris *et al.*, 2006a) ou (Degris, 2007) pour une présentation détaillée de l'algorithme et l'influence du paramètre $\tau_{\chi^2}$ sur l'apprentissage.

### 3.2. *Planification factorisée*

L'algorithme de planification que nous utilisons est une version incrémentale modifiée de l'algorithme SVI proposé par (Boutilier *et al.*, 2000). SVI est l'adaptation de *Value Iteration* aux représentations factorisées. Il est décrit à la figure 5.

SVI repose sur deux opérateurs : $Merge(\{T_1, ..., T_n\})$ et $Regress(Tree(V), a)$. $Merge$ produit un arbre unique qui résulte de la fusion des arbres $T_1, ..., T_n$ et dont le label des feuilles est calculé au moyen d'une *fonction de combinaison* des labels des feuilles impliquées dans la fusion des arbres originaux. $Regress$ produit $Tree(Q^V_a)$, qui représente la fonction de valeur d'action $Q^V_a$ étant donné la fonction de valeur $Tree(V)$. Il itère l'équation de Bellman pour tous les états en s'appuyant sur la repré-

Entrée: $\langle G_a|P_a\rangle, Tree(R)$ Sortie: $Tree(\pi^*)$

1) $Tree(V) \leftarrow Tree(R)$
2) Répéter jusqu'à l'arrêt :
   a) $\forall a \in A : Tree(Q_a^V) \leftarrow Regress(Tree(V), a)$
   b) $Tree(V) \leftarrow Merge(\{Tree(Q_a^V) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison).
3) $\forall a \in A : Tree(Q_a^V) \leftarrow Regress(Tree(V), a)$
4) $Tree(\pi) \leftarrow Merge(\{Tree(Q_a^V) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison et en plaçant les actions comme label des feuilles).
5) Renvoyer $Tree(\pi)$

**Figure 5.** *L'algorithme* SVI

sentation arborescente. Nous renvoyons à (Boutilier *et al.*, 2000) pour une description plus détaillée des opérateurs $Merge$ et $Regress$.

Utiliser SVI tel quel est possible, mais peu adapté pour deux raisons. Tout d'abord, SVI améliorerait la fonction de valeur jusqu'à convergence, malgré un modèle incomplet, donc convergerait à un coût élevé vers une fonction de valeur éventuellement erronée. Ensuite, la sortie de SVI est une politique gloutonne qui peut s'avérer inadaptée pour un problème dans lequel les fonctions de transition et de récompense sont inconnues. Dans un tel cadre, les agents doivent essayer des actions sous-optimales pour acquérir de l'information. Nous proposons donc une version incrémentale modifiée de SVI décrite sur la figure 6.

Entrée: $Tree(R_t), Tree(T_t), Tree(V_{t-1})$
Sortie: $Tree(V_t), \{\forall a \in A : Tree(Q_t(s,a))\}$

1) $\forall a \in A : Tree(Q_t(s,a)) \leftarrow Regress(Tree(V_{t-1}), a)$
2) $Tree(V_t) \leftarrow Merge(\{Tree(Q_t(s,a)) : \forall a \in A\})$ (en utilisant la maximisation des valeurs comme fonction de combinaison).
3) Renvoyer $\{Tree(V_t), \{\forall a \in A : Tree(Q_t(s,a))\}\}$

**Figure 6.** *L'algorithme IncSVI fondé sur* SVI

A chaque pas de temps, l'algorithme IncSVI effectue une itération de SVI et met à jour pour chaque action $a$ la représentation structurée de sa fonction de valeur $Tree(Q_t(s,a))$. La politique gloutonne est construite en recherchant la meilleure action dans l'état $s$ en fonction des valeurs de $Tree(Q_t(s,a))$. Cette information est ensuite utilisée par la phase d'action pour réaliser une politique $\epsilon$-*greedy* qui consiste à choisir une action au hasard avec une très faible probabilité $\epsilon$ et l'action de meilleure valeur dans le cas complémentaire (Sutton *et al.*, 1998).

La fonction de valeur $Tree(V_t)$ est calculée en fusionnant les fonctions de valeur d'action $\{Tree(Q_t(s,a)) : \forall a \in A\}$ (en utilisant la maximisation des valeurs comme

fonction de combinaison). $Tree(V_t)$ est ensuite réutilisé à l'instant $t+1$ pour calculer les fonctions de valeur d'action $Tree(Q_{t+1}(s,a))$. Si les modèles des transitions et des récompenses appris par l'agent sont stationnaires, $Tree(V_t)$ converge vers la fonction de valeur optimale $Tree(V^+)$ sous-tendue par la connaissance actuelle de l'agent (Boutilier *et al.*, 2000). Cependant, tant que les modèles appris par l'agent ne sont pas assez précis, $Tree(V^+)$ peut différer significativement de la fonction de valeur optimale du problème d'apprentissage par renforcement à résoudre.

## 4. Application à Counter-Strike©

Afin de valider l'approche SDYNA sur un problème réel, nous avons utilisé comme plate-forme de test le jeu vidéo Counter-Strike© [3], une extension du jeu Half-Life© [4] développée par Valve© [5], principalement à cause de sa popularité dans le milieu du jeu vidéo.

Par contraste avec des études de performance de SDYNA sur des problèmes académiques que nous avons publiées ailleurs (Degris *et al.*, 2006a; Degris *et al.*, 2006b; Degris *et al.*, 2008), les résultats que nous présentons dans cette section ont pour but de donner une illustration de l'applicabilité de SDYNA à un problème réel. Par conséquent, nos résultats sont qualitatifs plutôt que quantitatifs. Par cette expérimentation, notre but est de répondre à des questions telles que : dans quelle mesure peut-on utiliser SDYNA pour mettre au point l'IA d'un jeu du commerce ? dans quel cadre méthodologique une telle application de SDYNA est-elle possible ? les représentations construites par l'apprentissage sont-elles compréhensibles et/ou manipulables ?

### 4.1. *Description du jeu*

Counter-Strike© est un jeu de tir subjectif, ou *First-Person Shooter*, dans lequel deux équipes, des terroristes et des antiterroristes, s'affrontent dans des courtes parties de quelques minutes. La figure 7(a) illustre un exemple de vue à la première personne du joueur dans le jeu. Suivant la carte utilisée, les deux équipes ont des objectifs différents. Pour nos expériences, nous avons utilisé la carte *de_dust*, qui apparaît figure 8.

Sur cette carte, l'équipe terroriste apparaît au lieu marqué **T** (voir figure 8) et doit poser une bombe sur l'un des deux sites de bombe existants dans la carte (marqué **SA** et **SB** sur la carte de la figure 8). Les joueurs de l'équipe antiterroriste démarrent au lieu marqué **A** et doivent empêcher les joueurs de l'équipe terroriste de poser la bombe. Plusieurs conditions peuvent causer la fin de la partie en cours :

– le temps limite pour effectuer la mission a été dépassé ;

– la bombe a été posée par l'équipe terroriste puis a explosé ;

3. http ://counter-strike.net
4. http ://planethalflife.gamespy.com
5. http ://valvesoftware.com


(a) (b)

**Figure 7.** *Différentes captures d'écran du jeu Counter-Strike©. Figure a : vue du joueur dans le jeu. Figure b : un joueur de l'équipe terroriste installant la bombe sur le site de bombe A*

– la bombe a été posée par l'équipe terroriste puis a été désamorcée par l'équipe antiterroriste ;

– tous les membres d'une des deux équipes sont morts.

A la fin de chaque partie, chaque joueur de chaque équipe reçoit de l'argent en fonction de son score et de celui de son équipe et peut s'acheter, au début de la partie suivante, de nouvelles armes, de l'équipement supplémentaire (tel qu'un gilet pare-balles), des grenades (explosives, aveuglantes, ...) ou bien garder cet argent pour une prochaine partie.

Au cours de la partie, un joueur peut donc se déplacer (sur toute la carte), sélectionner une arme, tirer, ramasser des munitions, des armes ou la bombe, laissées au sol par d'autres joueurs et lancer des grenades. Une seule bombe existe lors d'une partie. Le joueur de l'équipe terroriste portant la bombe peut l'amorcer sur l'un des deux sites de bombe (figure 7(b)). Une fois que la bombe est posée, les joueurs de l'équipe antiterroriste peuvent la désamorcer avant que celle-ci explose.

Le joueur démarre une partie avec 100 points de vie. A chaque fois qu'il est touché (par un autre joueur, par une grenade ou bien par la bombe lorsqu'elle explose), il perd un certain nombre de points de vie (dépendant de l'endroit où il a été touché). Lorsque le nombre de points de vie tombe à 0, le joueur est mort et doit attendre le début de la partie suivante pour pouvoir jouer à nouveau. De plus, il n'est pas possible pour un joueur de récupérer des points de vie lors d'une partie. Un joueur qui n'est pas mort avant la fin d'une partie conserve son matériel pour la partie suivante.


**Figure 8.** *La carte* de_dust. *L'équipe terroriste doit déposer une bombe sur l'un des deux sites de bombe. L'équipe antiterroriste doit l'en empêcher*

L'équipe terroriste gagne soit parce que la bombe a explosé, soit parce que tous les joueurs de l'équipe antiterroriste ont été éliminés. L'équipe antiterroriste gagne soit parce que la bombe a été posée (par l'équipe adverse) puis désamorcée, soit parce que tous les joueurs de l'équipe terroriste ont été éliminés ou bien encore soit parce que le temps limite pour effectuer la mission est dépassé.

Enfin, à chaque fois qu'une équipe gagne, elle marque un point. Chaque joueur peut connaître aussi le nombre d'ennemis qu'il a tués et le nombre de fois qu'il est mort.

Dans ses premières versions, Counter-Strike© était un jeu qui se jouait exclusivement en réseau, c'est-à-dire que les joueurs pouvaient y jouer soit sur Internet, soit en constituant un réseau local[6]. Plusieurs extensions non officielles (développées par des volontaires) ont été proposées pour que l'ordinateur puisse contrôler des PNJ et qu'un joueur puisse jouer tout seul, contre l'ordinateur. Nous nous posons donc la question de savoir si SDYNA pourrait être une aide au développement de telles extensions, plus précisément : est-ce-qu'un agent SDYNA peut apprendre à contrôler un PNJ dans Counter-Strike© ?

6. Ce n'est plus le cas de la nouvelle version de Counter-Strike©, basée sur le moteur du jeu Half-Life© 2.

### 4.2. *Définition et formalisation du problème*

Afin de limiter le travail de développement important que nécessite une telle expérimentation, nous avons limité plusieurs paramètres dans le jeu. Notre but est d'apprendre à contrôler un PNJ de l'équipe terroriste et un PNJ de l'équipe antiterroriste. Un PNJ de l'équipe terroriste doit apprendre à aller poser la bombe et à savoir quand utiliser son arme. Nous n'avons pas eu le temps de développer le code nécessaire pour qu'un PNJ de l'équipe antiterroriste puisse apprendre à désamorcer la bombe. Par conséquent, un PNJ de cette équipe doit principalement apprendre à savoir quand utiliser son arme. De plus, les PNJ n'utilisent qu'une seule arme. Enfin, l'argent gagné lors de la mission précédente est utilisé de façon automatique pour acheter des munitions uniquement.

La structure de cette section suit plus ou moins la méthodologie que nous avons utilisée pour réaliser nos tests de SDYNA dans Counter-Strike©. Bien qu'étant différente, nous avons utilisé comme point de départ la formalisation du problème proposée par (Robert, 2005) pour le jeu Team Fortress Classic©. Section 4.2.1, nous commençons donc par définir les différentes récompenses que l'agent peut obtenir dans le jeu en fonction de ses objectifs. Section 4.2.2, nous définissons les perceptions de l'agent dans son environnement. Section 4.2.3, nous définissons les différentes actions qu'un agent peut exécuter dans l'environnement. Enfin, section 4.2.4, nous définissons la notion de pas de temps.

#### 4.2.1. *Définition des objectifs et récompenses associées*

À travers les récompenses que le PNJ peut obtenir dans le jeu, on définit ce que l'agent SDYNA doit apprendre. Or, un agent SDYNA contrôlant un PNJ de l'équipe terroriste doit remplir trois objectifs :

1) rester en vie ;

2) éliminer le plus possible des joueurs de l'équipe adverse ;

3) poser la bombe, s'il la possède.

On associe donc une récompense correspondant à chacun de ces objectifs :

1) -1 lorsque le PNJ est tué (0 sinon) ;

2) +10 lorsque le PNJ tue un adversaire (0 sinon) ;

3) +1 lorsque la bombe est posée (0 sinon).

Ces récompenses étant d'amplitude variable, cela revient à associer un poids relatif à chaque source de récompense et la modification de ces poids entraîne des modifications du comportement de l'agent. On procède de la même façon concernant l'agent SDYNA contrôlant un PNJ de l'équipe antiterroriste. Celui-ci a deux objectifs :

1) rester en vie

2) éliminer le plus possible des joueurs de l'équipe adverse.

Il obtient donc comme récompense :

1) -1 lorsque le PNJ est tué (0 sinon) ;

2) +10 lorsque le PNJ tue un adversaire (0 sinon).

Contrairement aux autres récompenses, la récompense obtenue par le PNJ de l'équipe terroriste lorsque la bombe est posée ne correspond pas au score obtenu par son équipe. En effet, les récompenses associées à « rester en vie » et « éliminer le plus possible des joueurs de l'équipe adverse » correspondent directement aux scores du joueur, respectivement le nombre de fois que le PNJ est mort et le nombre de fois qu'il a tué un adversaire.

Il aurait été plus difficile de donner une récompense associée aux scores de l'équipe puisque, comme nous l'avons décrit lors de la section 4.1, une équipe peut gagner pour plusieurs raisons différentes : lorsque tous les joueurs de l'équipe adverse ont été éliminés ou bien lorsque la mission est remplie. Pour que l'apprentissage d'une telle récompense soit possible, nous aurions été obligé d'ajouter un grand nombre de variables aléatoires dans l'espace d'état, par exemple pour indiquer le nombre de joueurs restant dans l'équipe adverse. Nous avons donc simplifié le problème en attribuant une récompense pour un PNJ de l'équipe terroriste lorsque la bombe était posée.

#### 4.2.2. *Définition de l'ensemble d'états*

Afin de pouvoir représenter le problème sous la forme d'un FMDP, nous décrivons l'espace d'état des agents SDYNA par un ensemble de variables aléatoires. Chaque variable aléatoire correspond à une perception de l'agent de son environnement, c'est-à-dire une observation concernant soit l'état courant du jeu, soit l'état personnel du PNJ. Dans la suite, nous utiliserons le mot « perception » plutôt que « variable aléatoire ».

| Perception | Signification | Valeurs possibles | Qui |
|---|---|---|---|
| *Current position* | position courante dans la carte | SiteA, SiteB SiteT, SiteC, Ailleurs | tous |
| *Has been shooted* | le PNJ vient-il d'être touché ? | Oui ou Non | tous |
| *Ammunition* | reste-t-il des munitions ? | Oui ou Non | tous |
| *Health* | santé du PNJ | Mort ou Vivant | tous |
| *Visible target* | le PNJ voit-il une cible ? | Oui ou Non | tous |
| *Carrying the bomb* | le PNJ possède-t-il la bombe ? | Oui ou Non | terroristes |
| *Bomb status* | la bombe est-elle posée ? | Posée, Non posée ou Peut être posée | terroristes |

**Tableau 1.** *Perceptions des* PNJ. *SiteA et SiteB désignent les sites de la bombe A et B respectivement, SiteT et SiteC les point de départ des terroristes et des antiterroristes, respectivement*

Les perceptions des PNJ des deux équipes dans le jeu sont données dans la table 4.2.2. Pour que les PNJ de l'équipe terroriste puissent apprendre à poser la bombe, nous leur rajoutons deux perceptions spécifiques.

Ainsi, la taille des espaces d'état sont de 80 et 480 états respectivement pour les PNJ de l'équipe des antiterroristes et des terroristes. Pour les PNJ de l'équipe des terroristes, certains états sont impossibles. Par exemple, la perception de l'état de la bombe ne peut pas être égale à « Peut être posée » lorsque le PNJ n'est pas sur un site de bombe.

Par conséquent, l'espace d'état du problème que nous tentons de résoudre est de petite taille par rapport à d'autres problèmes résolus avec SDYNA (Degris *et al.*, 2006a; Degris, 2007). Cependant, nous verrons section 6 qu'il présente plusieurs difficultés supplémentaires, le rendant ainsi très informatif concernant la robustesse de l'approche SDYNA face à un problème réel.

#### 4.2.3. *Définition de l'ensemble d'actions*

On définit les différentes actions d'un PNJ dans le jeu dans la table 4.2.3. Toutes ces actions sont des actions de haut niveau supposant l'existence d'actions bas niveau afin d'être réalisées. Par exemple, pour exécuter une action de type « Aller à », nous supposons l'existence d'un graphe représentant la carte que nous utilisons avec un algorithme $A^*$ afin de planifier la trajectoire de l'agent pour qu'il puisse atteindre son objectif. De même, pour l'action « *Shoot* », nous supposons l'existence d'un algorithme permettant de sélectionner la cible et d'orienter le PNJ vers celle-ci, puis de tirer. Ces actions de plus bas niveau ont été codées manuellement.

| Action | Signification | Qui |
|---|---|---|
| *Shoot* | tire sur un adversaire visible ou dans le vide | tous |
| *Go to the terrorist spawn* | se déplace vers SiteT | tous |
| *Go to the anti-terrorist spawn* | se déplace vers SiteC | tous |
| *Go to bomb site A* | se déplace vers SiteA | tous |
| *Go to bomb site B* | se déplace vers SiteB | tous |
| *Plant the bomb* | arme la bombe si possible | terroristes |

**Tableau 2.** *Actions des* PNJ

#### 4.2.4. *Définition des pas de temps*

Un problème important dans la formalisation est la définition d'un pas de temps pour l'apprentissage. Ainsi, un PNJ doit pouvoir à la fois exécuter des actions de courte durée – par exemple, l'exécution de l'action « *Shoot* » dure une ou deux secondes – et exécuter des actions de longue durée – par exemple, l'exécution d'une action de déplacement peut durer plusieurs dizaines de secondes – et, enfin, avoir des réflexes – par exemple, lorsqu'il détecte une cible potentielle, son temps de réaction doit être autour de 100ms. De plus, l'état du PNJ et l'action qu'il effectue sont rafraîchis par le serveur environ toutes les 10ms (cette fréquence n'est pas constante et dépend étroitement de la charge du serveur).

Il est important de noter que ces différences d'échelle dans le temps pose un problème uniquement pour l'apprentissage et non pour l'exécution d'une politique de

l'agent qui serait déjà connue. En effet, nous rappelons qu'un agent SDYNA apprend à partir d'une observation $\langle s, a, s', r\rangle$ de l'environnement, c'est-à-dire une transition entre un état précédent $s$ et un état courant $s'$, après avoir exécuté l'action $a$ et reçu la récompense $r$. Pour un apprentissage tel que celui de SDYNA, la difficulté concerne donc la détection de la fin de l'exécution de l'action $a$.

Pour cela, nous avons utilisé une heuristique simple : la fin d'une action est définie comme étant soit un changement de l'état du PNJ (c'est-à-dire la modification de la valeur de l'une de ses perceptions), soit après un temps maximum, $T_{max} = 750$ rafraîchissements, pour lequel l'état de l'agent n'a pas changé. La valeur $T_{max}$ a été fixée en fonction du temps nécessaire (approximatif) pour effectuer la plus longue action possible. Dans notre cas, nous avons compté le nombre de rafraîchissements exécutés pendant que l'agent part du point de départ de l'équipe terroriste et arrive au point de départ de l'équipe antiterroriste.

### 4.3. *Mise en œuvre de SDYNA*

Pour pouvoir mettre en œuvre SDYNA pour la résolution du problème que nous avons défini dans le jeu Counter-Strike©, il est nécessaire de prendre en compte le fait que les fonctions de récompense sont stochastiques, contrairement aux instances précédentes de SDYNA (Degris *et al.*, 2006a). Par exemple, pour la même perception « Le PNJ voit une cible » et la même action « Shoot », la récompense obtenue change si la cible est morte, si le PNJ est mort (il s'est fait toucher par sa cible ou un autre joueur), si la cible disparaît du champ de vision ou bien encore s'il ne reste plus de munitions.

Afin de pouvoir apprendre des fonctions stochastiques pour construire une représentation de la fonction de transition d'un FMDP représentant le problème, (Degris *et al.*, 2006a) utilise un pré-élagage basé sur la mesure d'information pour des valeurs symboliques. Nous utilisons une technique similaire pour l'apprentissage des fonctions de récompense du problème Counter-Strike©. Ainsi, un nœud de décision doit satisfaire deux critères pour être installé :

1) un critère évaluant une certitude : un nœud de décision testant l'attribut $\mathcal{V}$ est installé s'il existe au moins 4 exemples pour chaque valeur de $\mathcal{V}$ dans le domaine correspondant à la branche de l'arbre dans laquelle on se situe ;

2) un critère évaluant l'approximation de l'apprentissage : un test est installé si au moins l'une de ses branches est pure. Ce critère suppose qu'il n'existe pas de bruit dans la récompense obtenue par l'agent, il n'est donc pas adapté à l'apprentissage de fonctions stochastiques en général.

L'impact du facteur d'élagage sur l'apprentissage a été étudié sur des problèmes académiques dans (Degris *et al.*, 2006a) et dans (Degris, 2007), on retrouve les mêmes phénomènes ici donc nous ne les détaillerons pas davantage.

Finalement, l'agent SDYNA que nous utilisons est composé de l'algorithme d'apprentissage incrémental des modèles des fonctions de transition et de récompense décrit à la section 3.1 pour la mise à jour du FMDP, de l'algorithme IncSVI décrit à la section 3.2 pour la planification et de l'algorithme $\epsilon$-*greedy* pour la sélection de l'action. Nous utilisons une valeur de $\gamma = 0.99$, un seuil pour l'apprentissage des distributions de probabilités conditionnelles de $\tau_{\chi^2} = 30$ et $\epsilon = 0.1$ pour l'algorithme d'exploration $\epsilon$-*greedy*.

## 5. Résultats

Le protocole expérimental que nous utilisons est le suivant. L'équipe des terroristes est composée de deux PNJ utilisant chacun le même agent SDYNA décrit ci-dessus. Ainsi, l'apprentissage est partagé par les deux PNJ de l'équipe. L'équipe des antiterroristes est composée d'un seul joueur utilisant lui aussi l'agent SDYNA décrit ci-dessus. L'avantage est ainsi donné aux PNJ de l'équipe terroriste. Une partie du jeu dure cinq minutes. Enfin, les résultats présentés dans cette section ont été obtenus après un temps d'apprentissage de 2h08 pour l'agent SDYNA contrôlant les PNJ de l'équipe des terroristes et de 2h38 pour l'agent SDYNA contrôlant les PNJ de l'équipe des antiterroristes. Ces durées d'expérience correspondent à la fois au temps réel et au temps écoulé dans le jeu car nous n'avions pas la possibilité d'accélérer le serveur de jeu.

Cette section décrit plusieurs fonctions du FMDP obtenues par les agents lors de l'apprentissage et de la planification. Nous commençons par décrire, section 5.1, les fonctions de récompense. Section 5.2, nous décrirons plusieurs distributions de probabilités conditionnelles extraites de la fonction de transition. Enfin, nous décrirons plusieurs politiques gloutonnes construites par la planification dans la section 5.3.

### 5.1. *Fonction de récompense*

Le but de l'apprentissage supervisé effectué par SDYNA est la construction d'un FMDP représentant le problème à résoudre. Concernant les fonctions de récompense de ce FMDP, l'apprentissage construit une représentation (sous la forme d'un arbre de décision dans notre cas) qui, en fonction des perceptions de l'agent et de l'action qu'il exécute, associe la récompense obtenue par l'agent.

La figure 9 montre le résultat d'un tel apprentissage pour les PNJ de l'équipe terroriste pour chaque objectif : « poser la bombe », « rester en vie » et « éliminer le plus possible des joueurs de l'équipe adverse » illustrés respectivement dans les figures 9(a), 9(b), 9(c).

Ainsi, on peut constater que l'objectif « éliminer le plus possible des joueurs de l'équipe adverse » associant une récompense de +10 lorsque le PNJ tue un adversaire et 0 sinon, a été représenté par l'apprentissage en fonction des perceptions de l'agent et de l'action qu'il exécute. Plus précisément, l'agent a appris qu'il reçoit une récom-

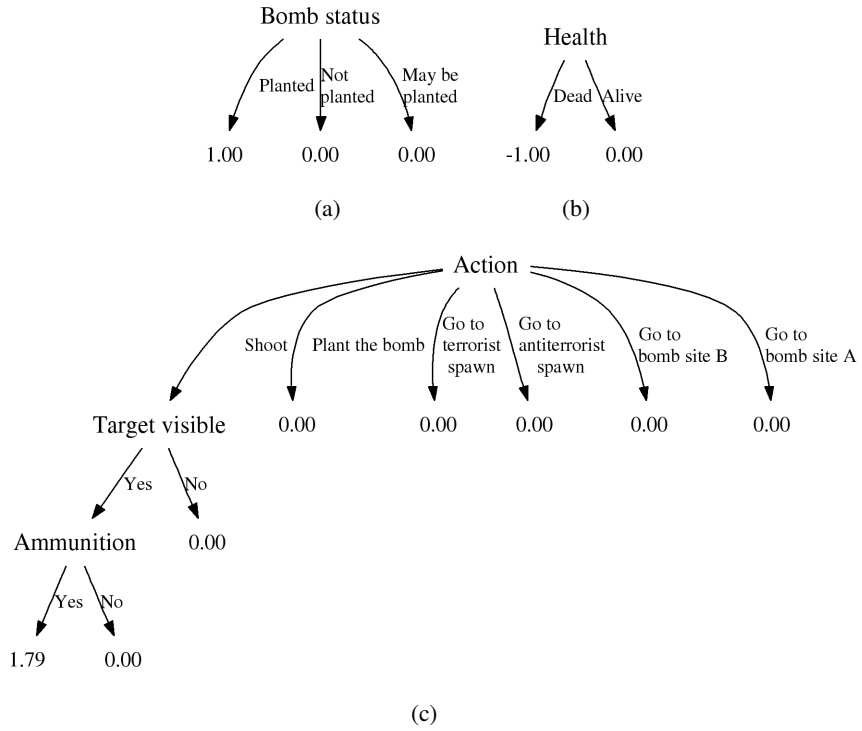



**Figure 9.** *Résultat de l'apprentissage d'un* PNJ *de l'équipe des joueurs terroristes pour les récompenses associées aux objectifs « poser la bombe » (figure a), « rester en vie » (figure b) et « éliminer le plus possible des joueurs de l'équipe adverse » (figure c). Dans la figure c, la feuille marquée 1.79 indique que l'agent estime que, s'il exécute l'action Shoot, s'il voit une cible et s'il a des munitions, il obtiendra une récompense de 1.79*

pense non nulle (différente de +10 puisqu'il ne tue pas sa cible systématiquement) lorsqu'il exécute l'action *Shoot*, qu'il voit une cible et qu'il lui reste des munitions.

Un apprentissage similaire a été effectué pour les autres objectifs du PNJ. Pour l'objectif « rester en vie », l'apprentissage a associé une récompense de -1 lorsque la perception de l'état de santé du PNJ indique que celui-ci est mort. Enfin, pour l'objectif « poser la bombe », l'apprentissage a associé une récompense de +1 lorsque la perception du statut de la bombe indique que celle-ci est posée. On remarque que, pour ces deux objectifs, la récompense obtenue par l'agent ne dépend pas d'une action en particulier mais plutôt de son état courant et de celui de la partie. Il faut avoir à l'esprit le fait que c'est l'algorithme d'induction des arbres de décision qui a sélectionné les variables pertinentes pour identifier les bonnes dépendances.

### 5.2. *Fonction de transition*

Le but de l'apprentissage supervisé effectué par SDYNA est la construction d'un FMDP représentant le problème à résoudre. Concernant les fonctions de transition de ce FMDP, l'apprentissage doit construire une représentation (sous la forme d'un arbre de décision dans notre cas) d'une distribution de probabilités conditionnelle pour chaque perception et pour chaque action de l'agent. En d'autres termes, l'apprentissage construit une fonction indiquant pour chaque perception quelle sera la valeur de cette perception au prochain pas de temps, en fonction des perceptions et de l'action de l'agent au pas de temps courant. L'apprentissage du modèle des transitions construit un arbre par perception, et non un arbre par perception et par action. Cette section présente quelques exemples caractéristiques de distributions de probabilités conditionnelles construites par l'apprentissage pour les PNJ appartenant à l'équipe des terroristes.

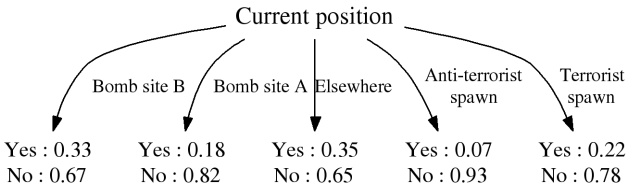



**Figure 10.** *Résultat de l'apprentissage d'un* PNJ *de l'équipe des joueurs terroristes pour la distribution de probabilités conditionnelle de la variable « Visible target ». La feuille la plus à droite indique que l'agent estime qu'il a une probabilité de 0.22 de voir une cible au prochain pas de temps (et une probabilité de 0.78 de ne pas en voir) lorsque sa position courante est le point de départ des terroristes*

La figure 10 montre le résultat de l'apprentissage pour la perception « *Visible target* ». L'arbre construit n'est composé que d'un seul nœud de décision testant la valeur de la perception indiquant la position courante de l'agent. D'après l'apprentissage, on peut donc voir que le fait de voir une cible au prochain pas de temps ne dépend pas de l'action exécutée par l'agent. De plus, on peut lire que l'agent estime qu'il est plus probable de voir une cible au prochain pas de temps lorsque sa position courante est « Ailleurs » (avec une probabilité de 0.35) ou « Site de bombe B » (avec une probabilité de 0.33) plutôt que « Point de départ des antiterroristes » (probabilité de 0.07).

La figure 11 montre le résultat de l'apprentissage pour la perception « *Ammunition* ». L'arbre construit est composé de deux nœuds de décision testant la valeur de la perception au pas de temps courant et l'action exécutée par l'agent. On remarque que, lorsque l'agent n'a plus de munitions, il a une probabilité de 1.0 de ne pas en avoir au prochain pas de temps, représentant ainsi le fait qu'il n'est pas possible dans l'expérience que nous avons menée de récupérer des munitions. Lorsque l'agent a des munitions, on remarque aussi que l'action *Shoot* est la seule action ne garantissant pas qu'il en aura au prochain pas de temps (avec une probabilité de 0.3 que l'agent

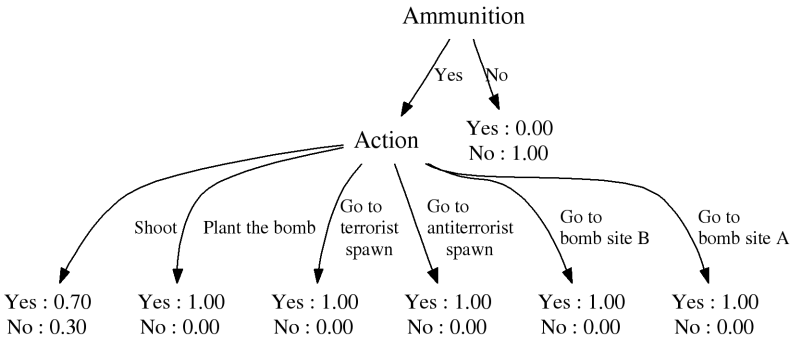



**Figure 11.** *Résultat de l'apprentissage d'un* PNJ *de l'équipe des joueurs terroristes pour la distribution de probabilités conditionnelle de la variable « Ammunition ». La feuille la plus à gauche indique que l'agent estime qu'il a une probabilité de 0.3 de ne plus avoir de munitions au prochain pas de temps (et une probabilité de 0.7 d'en avoir encore) lorsqu'il a des munitions et qu'il exécute l'action Shoot*

n'ait plus de munitions au prochain pas de temps), traduisant ainsi le fait qu'exécuter l'action *Shoot* coûte des munitions (ce qui n'est pas le cas des autres actions).

La figure 12 montre le résultat de l'apprentissage pour la perception « *Bomb status* », une variable représentant l'état courant de l'environnement plutôt que l'état interne du PNJ. On remarque en premier lieu que, lorsque la bombe est posée, il y a une probabilité de 1.0 qu'elle soit posée au prochain pas de temps, illustrant ainsi que la bombe n'a jamais été désamorcée (puisque les antiterroristes n'en ont pas la capacité).

L'arbre indique aussi que pour qu'un agent puisse poser la bombe, il est nécessaire qu'elle ne soit pas posée, que sa position courante soit sur un site de bombe (A ou B) et qu'il possède la bombe. Lorsque la bombe peut être posée (seul l'agent possédant la bombe peut obtenir une telle valeur pour la perception « *Bomb status* »), on observe que seule l'action *Plant the bomb* est associée à une probabilité non nulle que la bombe soit posée au prochain pas de temps.

Lorsque la bombe n'est pas posée et que la position courante d'un agent est, par exemple, « Ailleurs », on note que la probabilité que la bombe soit posée au prochain pas de temps est non nulle. La perception « *Bomb status* » représentant un état de l'environnement et non de l'agent, cette probabilité indique qu'un autre PNJ de l'équipe des terroristes a posé la bombe.

Enfin, on remarque que l'action exécutée par l'agent est seulement testée lorsque la perception « *Bomb status* » de l'agent est égale à « Peut être posée ». Ainsi, le reste de l'arbre (correspondant aux autres valeurs de la perception) est commun à toutes les actions.

### 5.3. *Politiques gloutonnes*

Une fois que l'apprentissage a défini les fonctions de transition et de récompense du FMDP représentant le problème, il est possible d'utiliser la planification pour calculer une politique pour l'agent. Les résultats présentés dans cette section illustrent des politiques gloutonnes (c'est-à-dire sélectionnant la ou les meilleures actions) calculées à partir du FMDP appris par l'agent SDYNA contrôlant les PNJ de l'équipe terroriste. Au cours de l'expérimentation, nous avons utilisé l'algorithme incrémental de planification IncSVI pour déterminer l'action exécutée par les PNJ dans le jeu. Cet algorithme ne calcule pas de représentation explicite de la politique. Afin de pouvoir les montrer dans cette section, nous avons calculé ces politiques en utilisant l'algorithme SVI avec le FMDP construit par l'apprentissage. Une fois la politique calculée, l'arbre de décision la représentant est réorganisé pour une meilleure lisibilité. Une feuille marquée « Pas de sélection » dans un arbre de décision représentant une politique signifie que toutes les actions possibles sont considérées comme équivalentes par l'agent.

La figure 13 représente la politique gloutonne calculée à partir des fonctions de transition et de récompense construites au cours de l'apprentissage des PNJ de l'équipe des terroristes. Cette politique a pour but de maximiser l'objectif « poser la bombe ». Les récompenses associées aux deux autres objectifs, « rester en vie » et « éliminer le plus possible des joueurs de l'équipe adverse » sont ignorées.

En premier lieu, on constate que lorsque le PNJ peut poser la bombe, la meilleure action est *Plant the bomb*. Une fois que la bombe a été posée, l'objectif a été atteint et aucune action n'est considérée comme étant meilleure qu'une autre. Lorsque la bombe n'a pas été posée, la meilleure action dépend de la position courante du PNJ. En effet, si le PNJ est sur un site de bombe, alors il considère que toutes les actions n'ayant aucun effet sur sa position, c'est-à-dire les actions *Shoot* et *Plant the bomb* sont équivalentes. Au contraire, lorsque le PNJ se situe Ailleurs ou au Point de départ des antiterroristes, la meilleure action est *Go to bomb site A*. Ce n'est pas le cas lorsque le PNJ est à son point de départ, auquel cas toutes les actions sont considérées comme équivalentes.

La figure 14 représente une autre politique gloutonne calculée à partir des fonctions de transition et de récompense construites au cours de l'apprentissage des PNJ de l'équipe des terroristes. Cette politique a pour but de maximiser l'objectif « éliminer le plus possible des joueurs de l'équipe adverse ». Les récompenses associées aux deux autres objectifs, « rester en vie » et « poser la bombe » sont ignorées.

On observe que, lorsque le PNJ n'a plus de munitions, alors toutes les actions sont considérées comme équivalentes. Au contraire, lorsque le PNJ a des munitions et qu'il voit une cible, alors il considère que la meilleure action est *Shoot*. Dans le cas où le PNJ a des munitions et qu'il ne voit pas de cible, la meilleure action dépend de sa position courante. Nous avons vu, lors de la description de la distribution de probabilités conditionnelle de la perception *Visible target* dans la figure 10 (page 239), que la probabilité de voir une cible au pas de temps suivant était plus importante lorsque la position courante du PNJ était Ailleurs.

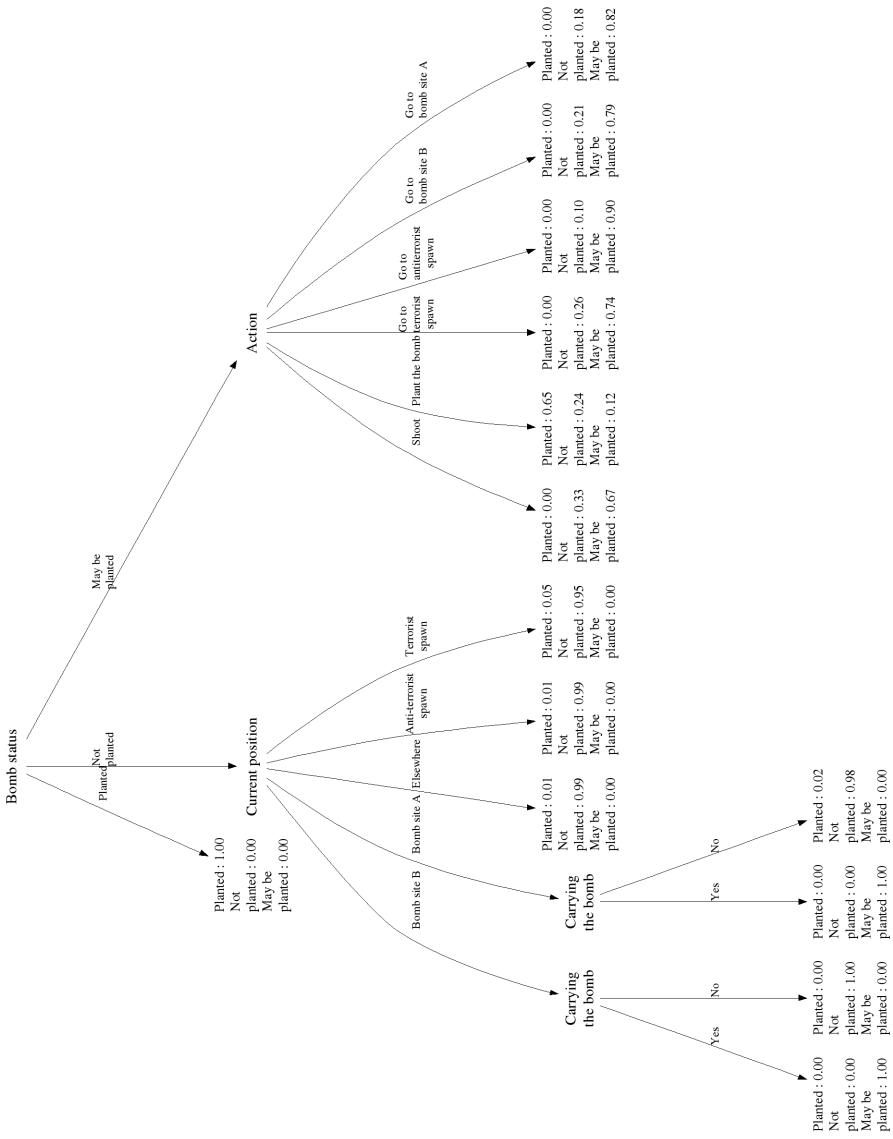



**Figure 12.** *Résultat de l'apprentissage d'un* PNJ *de l'équipe des joueurs terroristes pour la distribution de probabilités conditionnelle de la variable « Bomb status ». La feuille la plus à droite indique que l'agent estime qu'au prochain pas de temps, lorsque la bombe peut être posée (par cet agent) et que l'agent exécute l'action Go to bomb site A, il y a une probabilité nulle que la bombe soit posée, une probabilité de 0.18 que la bombe ne soit pas posée et une probabilité de 0.82 qu'elle puisse être posée par cet agent*

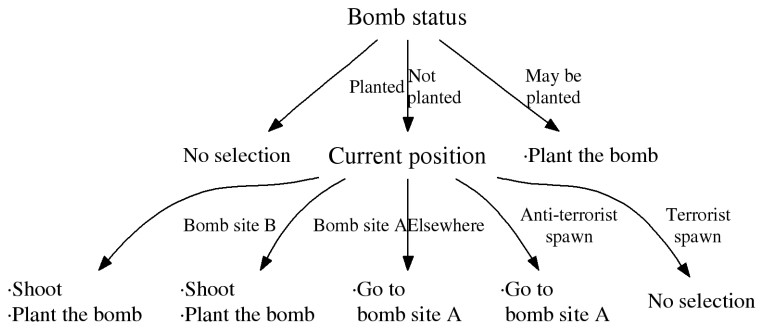



**Figure 13.** *Résultat du calcul de la politique d'un* PNJ *de l'équipe des joueurs terroristes pour maximiser la récompense associée à l'objectif « poser la bombe » à partir de l'apprentissage. La feuille la plus à gauche signifie que lorsque la bombe n'est pas posée et la position courante du* PNJ *est le site de bombe B, les deux meilleures actions sont Shoot et Plant the bomb*

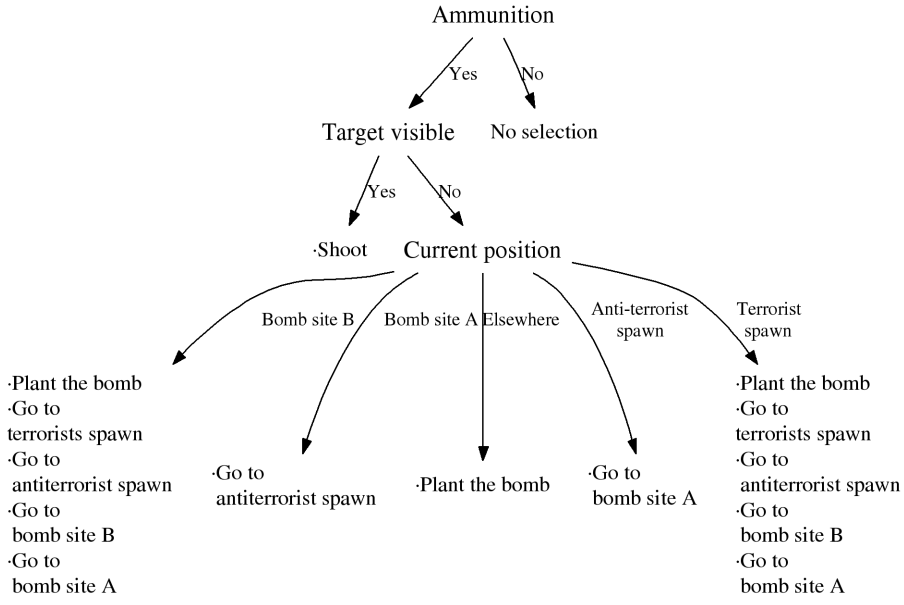



**Figure 14.** *Résultat du calcul de la politique d'un* PNJ *de l'équipe des joueurs terroristes pour maximiser la récompense associée à l'objectif « éliminer le plus possible des joueurs de l'équipe adverse » à partir de l'apprentissage*

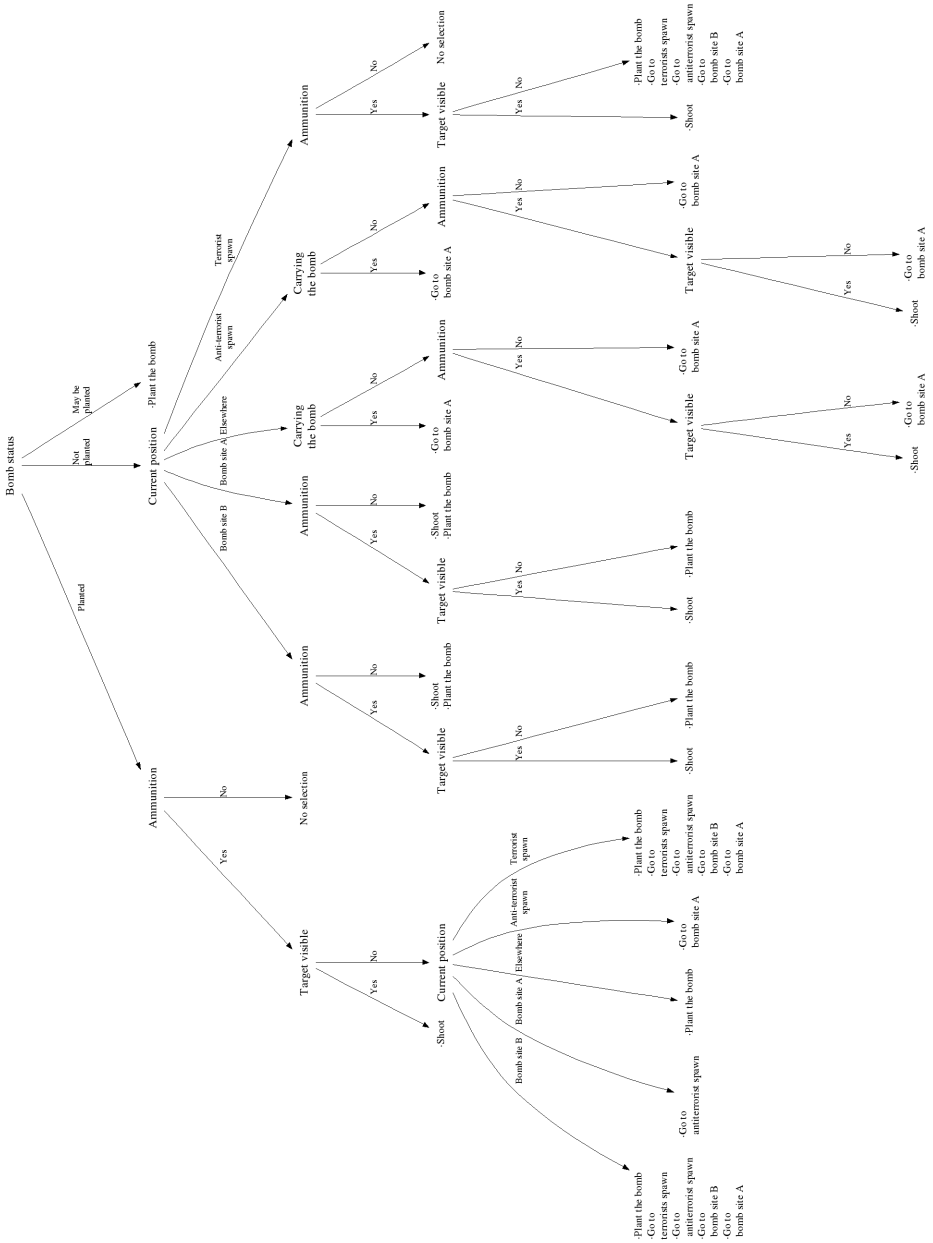



**Figure 15.** *Résultat du calcul de la politique d'un* PNJ *de l'équipe des joueurs terroristes pour maximiser la somme de l'ensemble des récompenses associées à l'ensemble des objectifs du* PNJ

C'est la raison pour laquelle, lorsque le PNJ a des munitions mais ne voit pas de cible, la meilleure action estimée est *Plant the bomb* lorsque l'agent est Ailleurs, c'est-à-dire la seule action n'ayant pas d'effet ni sur la position courante de l'agent, ni sur ses munitions. Pour les lieux Site de bombe B et Point de départ des terroristes, toutes les actions sont considérées équivalentes, sauf l'action *Shoot* qui a un effet sur les munitions. Enfin, pour les lieux Point de départ des antiterroristes et Site de bombe A, la probabilité de voir une cible au prochain pas de temps n'est pas suffisante et la meilleure action correspond à la probabilité la plus élevée d'être Ailleurs au prochain pas de temps.

La figure 15 représente la politique gloutonne calculée à partir des fonctions de transition et en considérant l'ensemble des fonctions de récompense construites au cours de l'apprentissage des PNJ de l'équipe des terroristes. Cette politique a pour but de maximiser la somme des récompenses associées à chacun des objectifs du PNJ dans le jeu.

On remarque que l'on retrouve les mêmes meilleures actions que celles observées dans les politiques précédentes. Un premier exemple est lorsque le PNJ peut poser la bombe, alors la meilleure action est *Plant the bomb*. Un deuxième exemple est le fait que l'on retrouve à plusieurs endroits dans l'arbre le fait que lorsque le PNJ a une cible et qu'il a des munitions, la meilleure action sélectionnée est *Shoot*. On peut remarquer que la meilleure action considérée par l'agent est *Go to bomb site A* lorsque la bombe n'est pas posée, que le PNJ est Ailleurs et qu'il est en possession de la bombe. La perception *Visible target* n'étant pas testée, l'agent préfère donc aller poser la bombe plutôt que tirer sur un adversaire s'il en voyait un.

Au bout du compte, au bout de 2h d'apprentissage, on dispose de PNJ dont le comportement est globalement satisfaisant, comme en témoigne qualitativement la vidéo disponible à l'adresse : http ://animatlab.lip6.fr/KodabotFr.

## 6. Discussion

Dans cette section, nous commençons par mettre en évidence les leçons que nous pouvons tirer localement de nos expériences dans le cadre particulier de Counter-Strike©. Puis nous évaluons la portée de nos résultats en prenant du recul sur les difficultés de cette application par rapport à des *benchmarks* du domaine. Enfin, nous discutons de l'impact potentiel des travaux que nous avons présentés sur la mise au point d'IA pour les jeux vidéo de façon plus générale.

### 6.1. *Evaluation de nos résultats*

Nos résultats montrent que l'agent SDYNA a construit des représentations pertinentes des fonctions de transition et de récompense sous une forme directement intelligibles par un utilisateur. En outre, les algorithmes de planification permettent de construire des représentations intelligibles des politiques gloutonnes de l'agent et,

d'autre part, l'apprentissage s'est révélé suffisamment pertinent pour que la politique de l'agent soit adaptée au problème que celui-ci doit résoudre.

De plus, l'arbre de décision représentant la perception « *Bomb status* » illustre le gain de place et de lisibilité réalisé lorsque la fonction de transition du FMDP représentant le problème n'est composée que d'un arbre par variable. En effet, si un arbre par action et par variable avait été construit, la partie de l'arbre non dépendante de l'action aurait été reproduite pour toutes les actions possibles dans le problème, rendant sa lecture plus difficile et son encombrement mémoire plus important.

On remarque aussi qu'une représentation sous la forme d'arbre de décision, en plus de la lisibilité, permet de faire des économies importantes concernant les temps de calcul requis pour décider de la prochaine action du PNJ (lorsque l'apprentissage est arrêté). En effet, par exemple, lorsque le PNJ peut poser la bombe, le statut de la bombe est la seule perception évaluée pour déterminer l'action à réaliser par l'agent.

On notera que, lorsque l'apprentissage est incomplet, l'agent n'est pas capable de discriminer une action par rapport à une autre, comme c'est le cas pour la politique maximisant l'objectif « poser la bombe » lorsque l'agent est au point de départ des terroristes (voir figure 13). Dans ce cas, les actions seront choisies de façon aléatoire jusqu'à ce que l'une d'entre elles s'aèvre meilleure que les autres.

Enfin, on peut s'interroger sur la pertinence de l'apprentissage des fonctions de récompense. En effet, lors de la définition d'un problème d'apprentissage par renforcement, il est nécessaire de définir par programme les circonstances dans lesquelles des récompenses sont obtenues par l'agent, ce qui revient à définir les objectifs de l'agent dans son environnement. Par conséquent, on pourrait penser que les fonctions de récompense pourraient être données *a priori* à l'agent et que leur apprentissage est moins nécessaire que celui des fonctions de transition.

En fait, l'apprentissage de ces fonctions supprime une contrainte importante qui pèse sur la définition des objectifs. En effet, les fonctions qui servent à déterminer si des objectifs sont atteints ne s'expriment pas nécessairement en fonction des différentes perceptions de l'agent dans son environnement. En revanche, l'apprentissage permet à l'agent de construire les fonctions de récompense caractérisant les objectifs du problème à partir des corrélations qu'il observe entre ses perceptions et les récompenses qu'il obtient dans son environnement. L'apprentissage de la fonction de récompense permet donc de faire le lien entre la définition des objectifs telle qu'elle est programmée par l'utilisateur et la représentation de ces objectifs que l'agent peut constituer à partir de ses propres perceptions.

### 6.2. *Difficulté intrinsèque du problème*

A partir de la définition du problème du jeu Counter-Strike© tel que nous l'avons défini ci-dessus, nous pouvons souligner plusieurs de ses caractéristiques. Tout d'abord, il ne satisfait pas l'hypothèse de Markov. En effet, l'état suivant de l'agent

dépend non seulement de son état courant et de son action, mais aussi des actions de tous les autres agents.

Par ailleurs, la perception très grossière dont dispose l'agent pour des variables de localisation, de santé ou de munitions, par exemple, induisent des difficultés sur la construction du modèle des transitions. Par exemple, l'agent peut déterminer une probabilité globale pour que, lorsqu'il tire, la perception « *Ammunition* » passe de la valeur Oui à Non, mais cette transition dépend essentiellement du nombre de fois qu'il a tiré précédemment, information qui n'est pas disponible dans le modèle.

On peut aussi noter le bruit induit sur la représentation par le déroulement des actions de bas niveau. D'une part, certaines actions choisies par l'agent sont soumises à des paramètres de l'environnement empêchant leur bon déroulement lors de leur exécution par le PNJ. Par exemple, nous avons remarqué plusieurs fois que, lorsque deux PNJ se croisaient, ils pouvaient rester bloqués l'un contre l'autre, empêchant ainsi le déroulement correct des actions de navigation. D'autre part, les agents doivent apprendre en présence d'ennemis dont le comportement évolue au fil du temps.

Malgré toutes ces difficultés, SDYNA parvient à proposer un comportement convaincant sur ce problème, même s'il n'est pas optimal, comme en témoignent nos expériences. Cela tient au fait que l'essentiel des informations nécessaires pour que l'agent puisse choisir une action pertinente sont présentes dans l'ensemble des perceptions décrivant l'état courant $s$ de l'agent.

Ensuite, nous pouvons remarquer que les perceptions et les actions que nous avons définies ne sont pas dépendantes de la carte que nous utilisons. En effet, toutes les cartes du même type que *de_dust* (la carte que nous utilisons pour nos expériences) possèdent deux sites de bombe ainsi que deux points de départ pour chaque équipe. Par conséquent, l'apprentissage réalisé sur une carte est directement réutilisable sur une autre carte du même type dans le jeu Counter-Strike©. Cette propriété vient du fait que nous utilisons une représentation de haut niveau et abstraite de la carte qui ne fait appel à aucune information spécifique de celle-ci (comme par exemple sa topologie).

Enfin, il faut noter que, si nous avons donné une représentation du problème de relativement petite taille par rapport à la taille des problèmes académiques que SDYNA est capable de résoudre, c'est non seulement afin de bénéficier des propriétés d'abstraction que nous venons de souligner, mais aussi parce que la programmation de l'extraction de perceptions et de la réalisation d'actions dans Counter-Strike© s'est avérée très ardue et coûteuse en temps, si bien que nous nous sommes limités à l'essentiel. Le choix d'une autre plate-forme de jeu dotée d'une véritable interface pour le développement du comportement des PNJ nous aurait sans doute permis d'aborder des problèmes de beaucoup plus grande taille tels qu'en rencontrent des éditeurs de jeux, sujet que nous allons évoquer à présent.

### 6.3. *Impact potentiel sur la mise au point d'IA pour les jeux*

La qualité de l'IA pour les PNJ d'un jeu vidéo du commerce est un facteur important du succès du jeu parce qu'elle procure aux joueurs des défis intéressants (Scott, 2002; Robert, 2005). Dans la plupart des cas, la mise au point de l'architecture de décision de ces PNJ est un problème trop complexe pour qu'une méthode entièrement automatique reposant sur SDYNA puisse suffire à le résoudre. En effet, l'IA peut se trouver confrontée au traitement d'un nombre considérable de variables impliquant une échelle de complexité bien supérieure à celle qui est affrontée dans cet article (Christian, 2002; Orkin, 2004; Champandard, 2008). De plus, certaines de ces variables peuvent être continues, ce qui pose des problèmes spécifiques que nous n'avons pas abordés ici, il faut prendre en compte la variabilité de la stratégie de nombreux joueurs humains, etc. En revanche, le budget limité et les contraintes temporelles associées à cette phase font qu'une méthode semi-automatique fondée sur SDYNA peut être la bienvenue pour faciliter localement le travail des développeurs en charge de l'IA.

Dans ce contexte de mise au point d'une IA, avant la diffusion du jeu, les qualités de clarté et de lisibilité que nous avons mises en avant sont essentielles, comme le souligne (Ponsen *et al.*, 2007). Le programmeur responsable de l'IA peut ainsi produire des comportements de manière automatisée en utilisant de l'apprentissage hors ligne, puis intervenir manuellement sur la politique des agents s'il le souhaite.

Par ailleurs, dans le cadre du développement du jeu lui-même, une méthode telle que SDYNA devrait permettre d'obtenir à moindre frais des comportements qui peuvent servir à tester l'état courant du jeu. Une telle méthode permet au *game designer* d'avoir des retours rapidement, ce qui facilite par exemple la recherche de failles dans les jeux. Au travers de notre expérience avec Counter-Strike©, nous avons constaté que le fait d'obtenir rapidement une IA lisible nous permettait à la lecture des arbres obtenus de détecter des bugs dans les fonctions faisant le lien entre les perceptions et actions de l'agent et le moteur de la simulation dans laquelle évolue le PNJ. Notre outil ne sert alors plus seulement à accélérer la mise au point de l'IA, mais aussi celle du jeu lui-même.

Il faut toutefois reconnaître que SDYNA n'a pas été utilisé jusqu'à présent en contexte industriel dans un studio de développement, si bien que les pistes d'utilisation que nous proposons ci-dessus restent essentiellement spéculatives.

Le second cadre d'utilisation potentiel de nos techniques d'apprentissage pour l'IA des jeux vidéo est une utilisation en ligne, ce qui signifie que l'apprentissage reste activé pendant que les utilisateurs finaux jouent. Les avantages d'une telle utilisation sont multiples (Tozour, 2002; Andrade *et al.*, 2005). Cependant, nous n'avons pas testé nos outils pour ce second type d'usage. Malgré son intérêt, un tel usage suppose des méthodes d'évaluation à grande échelle, mobilisant de nombreux joueurs sur une durée importante, ce qui sortait du cadre de notre étude. De tels travaux restent donc à l'état de perspectives.

Par ailleurs, nous n'avons pas examiné l'applicabilité de notre travail à d'autres types de jeux impliquant un grand nombre d'agents et des environnements beaucoup plus vastes, tels que les MMORPG (Robert *et al.*, 2002) ou les RTS (Madeira, 2007).

## 7. Conclusion

Cette contribution avait pour but d'illustrer la mise en œuvre de SDYNA sur le problème consistant à contrôler un PNJ dans le jeu vidéo Counter-Strike©. Dans un premier temps, nous avons expliqué le déroulement du jeu. Dans un deuxième temps, nous avons montré comment on pouvait formaliser la mise au point d'une IA sous la forme d'un problème d'apprentissage par renforcement. Enfin, nous avons décrit l'instance de SDYNA que nous avons utilisée pour résoudre ce problème et valider celle-ci au cours d'une expérience d'environ 2h00 (temps réel et virtuel) dans le jeu vidéo.

Nos résultats sont essentiellement qualitatifs plutôt que quantitatifs, mais ils suffisent à montrer que, pour ce problème, notre méthode est capable d'apprendre une représentation pertinente du problème sous la forme d'un FMDP, celui-ci étant défini par les fonctions de récompense et de transition. De plus, nous avons montré que ces représentations pouvaient facilement être interprétées par un utilisateur, facilitant ainsi la compréhension du problème et du comportement de l'agent.

Nous avons aussi montré que le FMDP construit par l'apprentissage permettait, par l'utilisation d'un algorithme de planification, de construire une représentation explicite de la politique gloutonne de l'agent. Cette politique représente efficacement le comportement d'un PNJ dans le jeu. Nous avons constaté que SDYNA peut construire des politiques adaptées et intelligibles au problème posé dans le jeu vidéo et que la représentation arborescente permet de faire des économies de mémoire et de calcul en évitant la représentation et l'évaluation de perceptions inutiles dans certains contextes.

Nous avons enfin discuté de la possibilité d'appliquer ces travaux au développement de l'IA de jeux du commerce dans un cadre hors ligne, en nous concentrant sur le cas des jeux de tir en première personne.

Remerciements

Nous tenons à remercier les nombreuses personnes qui ont permis de tester SDYNA sur Counter-Strike© dont, en particulier, Nicolas Desprès, Jean-Christophe Dubus et Alix Mougenot. Nous remercions aussi les relecteurs qui nous ont permis par leurs remarques d'améliorer cet article.

## 8. Bibliographie

Andrade G., Ramalho G., Santana H., Corruble V., « Extending Reinforcement Learning to Provide Dynamic Game Balancing », *Proceedings of the 2005 IJCAI Workshop on Rea-*

*soning, Representation, and Learning in Computer Games*, Edinburgh, Scotland, p. 7–12, 2005.

Boutilier C., Dearden R., Goldszmidt M., « Exploiting Structure in Policy Construction », *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, p. 1104–1111, 1995.

Boutilier C., Dearden R., Goldszmidt M., « Stochastic Dynamic Programming with Factored Representations », *Artificial Intelligence*, vol. 121, n° 1, p. 49–107, 2000.

Buffet O., Sigaud O., *Processus décisionnels de Markov en intelligence artificielle (volume 1)*, Hermes-Lavoisier, 2008.

Champandard A., « Getting Started with Decision Making and Control Systems », *AI Game Programming Wisdom*, Charles River Media Inc., 2008.

Christian M., « A simple inference engine for a rule-based architecture », *AI Game Programming Wisdom*, Charles River Media Inc., 2002.

DasGupta A., *Asymptotic Theory of Statistics and Probability*, Springer, New York, 2008.

Dean T., Kanazawa K., « A Model for Reasoning about Persistence and Causation », *Computational Intelligence*, vol. 5, p. 142–150, 1989.

Degris T., *Apprentissage par Renforcement dans les Processus de Décision Markoviens Factorisés*, Thèse de Doctorat de l'Université Paris 6, 2007.

Degris T., Sigaud O., Wuillemin P.-H., « Chi-square Tests Driven Method for Learning the Structure of Factored MDPs », *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, Cambridge, MA, USA, p. 122–129, 2006a.

Degris T., Sigaud O., Wuillemin P.-H., « Learning the Structure of Factored Markov Decision Processes in Reinforcement Learning Problems », *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, USA, p. 257–264, 2006b.

Degris T., Sigaud O., Wuillemin P.-H., « Exploiting Additive Structure in Factored MDPs for Reinforcement Learning », *Proceedings EWRL2008, LNAI 5323*, Lille, France, p. 15–26, 2008.

Guestrin C., Koller D., Gearhart C., Kanodia N., « Generalizing Plans to New Environments in Relational MDPs », *International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

Hoey J., St-Aubin R., Hu A., Boutilier C., « SPUDD : Stochastic Planning using Decision Diagrams », *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, p. 279–288, 1999.

Laird J. E., van Lent M., « Human-level AI Killer Application : Interactive Computer Games », *Proceedings AAAI 2000*, AAAI Press / The MIT Press, p. 1171-1178, 2000.

Madeira C., *Agents adaptatifs dans les jeux de stratégie modernes : une approche fondée sur l'apprentissage par renforcement*, Thèse de Doctorat de l'Université Paris 6, France, 2007.

Orkin J., « Applying goal-oriented action planning to games », *AI Game Programming Wisdom 2*, Charles River Media Inc., 2004.

Ponsen M., Spronck P., Muñoz Avila H., Aha D. W., « Knowledge acquisition for adaptive game AI », *Science of Computer Programming*, vol. 67, n° 1, p. 59–75, 2007.

Puterman M. L., *Markov Decision Processes : Discrete Stochastic Dynamic Programming.*, John Wiley and Sons, New York, 1996.

Quinlan J. R., « Induction of Decision Trees », *Machine Learning*, vol. 1, n° 1, p. 81–106, 1986.

Robert G., *MHiCS, une Architecture de Sélection de l'Action Motivationnelle et Hiérarchique à Systèmes de Classeurs pour Personnages Non Joueurs Adaptatifs*, Thèse de Doctorat de l'Université Paris 6, France, 2005.

Robert G., Portier P., Guillot A., « Classifier Systems as 'Animat' Architectures for Action Selection in MMORPG », *Proceedings of Game-On 2002*, p. 121–125, 2002.

Schlimmer J., Fisher D., « A Case Study of Incremental Concept Induction », *Proceedings of the Fifth National Conference on Artificial Intelligence*, p. 496–501, 1986.

Scott B., « The illusion of Intelligence », *AI Game Programming Wisdom*, Charles River Media Inc., p. 16-20, 2002.

Sigaud O., *Comportements Adaptatifs pour des Agents dans des Environnements Informatiques Complexes*, Habilitation à Diriger des Recherches de l'Université Paris 6, 2004.

Sigaud O., « Les systèmes de classeurs : un état de l'art », *Revue d'Intelligence Artificielle*, vol. 21, p. 75-106, 2007.

Sigaud O., Butz M., Kozlova O., Meyer C., « Anticipatory Learning Classifier Systems and Factored Reinforcement Learning », *LNAI, Proceedings ABiALS 2008*, Springer, p. to appear, 2009.

Spronck P., Ponsen M., Sprinkhuizen-Kuyper I., Postma E., « Adaptive Game AI with Dynamic Scripting », *Machine Learning*, vol. 63, n° 3, p. 217–248, 2006.

Sutton R. S., « Integrated architectures for learning, planning, and reacting based on approximating dynamic programming », *Proceedings of the Seventh International Conference on Machine Learning*, San Mateo, CA. Morgan Kaufmann, p. 216–224, 1990.

Sutton R. S., Barto A. G., *Reinforcement Learning : An Introduction*, MIT Press, 1998.

Tozour P., « The Evolution of Game AI », *AI Game Programming Wisdom*, Charles River Media Inc., p. 3-15, 2002.

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com

**ANNEXE POUR LE SERVICE FABRICATION**
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
*RSTI - RIA - 23/2009. Jeux : modélisation et décision*

2. AUTEURS :
*Thomas Degris*[*] *— Olivier Sigaud*[**] *— Pierre-Henri Wuillemin*[***]

3. TITRE DE L'ARTICLE :
*Apprentissage par renforcement factorisé pour le comportement de personnages non joueurs*

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
*Apprentissage du comportement de* PNJ

5. DATE DE CETTE VERSION :
*26 mars 2009*

6. COORDONNÉES DES AUTEURS :

– adresse postale :

[*] 3-34 Athabasca Hall — Department of Computing Science — University of Alberta — Edmonton, AB, Canada, T6G 2E8

degris@cs.ualberta.ca

[**] Institut des Systèmes Intelligents et de Robotique — Université Pierre et Marie Curie - Paris 6, CNRS UMR 7222 — Pyramide - Tour 55, Boîte courrier 173 — 4 place Jussieu, F75252 Paris Cedex 05

olivier.sigaud@upmc.fr

[***] Laboratoire d'Informatique de Paris 6 — Université Pierre et Marie Curie - Paris 6, CNRS UMR 7606 — 4 place Jussieu, F75252 Paris Cedex 05

pierre-henri.wuillemin@lip6.fr

– téléphone : 01 44 27 88 53

– télécopie : 01 46 54 72 99

– e-mail : olivier.sigaud@isir.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
LaTeX, avec le fichier de style `article-hermes.cls`,
version 1.25 du 03/07/2006.

8. FORMULAIRE DE COPYRIGHT :
Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
`http://www.revuesonline.com`