# Utilisation des modèles de semi-Markov cachés à des fins d'amélioration d'intelligences artificielles dans les jeux vidéo

Cédric Beaulac

Université du Québec à Montréal

15 décembre 2013

# Table des matières

# 1 Introduction

Dans la majorité des jeux vidéos compétitifs, il est possible, en tant que joueur, d'affronter des intelligences artificielles dans l'objectif de devenir plus performant. Par contre, pour réellement améliorer ces performances dans l'objectif d'affronter de vrais joueurs, les *bots* (terme souvent utilisé pour décrire une intelligence artificielle remplaçant un joueur usuellement humain) doivent être programmés pour réagir comme des êtres humains, performer comme des joueurs professionnels et agir le plus possible comme un *gamer* lorsqu'il est dans l'inconnu et ce sans tricher. L'idée ici est d'utiliser les chaînes de Markov cachées pour permettre aux intelligences artificielles d'estimer les probabilités des diverses positions possibles de son adversaire de manière réaliste.

En tant qu'intelligence artificielle, nous verrons ici chaque position potentielle de l'adversaire comme étant les états de notre modèle de Markov et notre objectif est de déterminer dans quel état se trouve le joueur. La probabilité de passer d'un état à un autre dépend bien entendu de l'état dans lequel l'adversaire se trouve. Nous considérons que le système est caché car nous ne connaissons pas la position du joueur, mais nous avons comme observation certaines positions à laquelle il ne se trouve pas. Nous utiliserons ici des modèles de semi-markov cachées car ceux-ci nous permettent de considérer que l'utilisateur du jeu, notre adversaire, restera un certain nombre de temps aléatoire à certaines positions. Deux types de jeux seront principalement étudiés dans ce papier, soit les FPS, des jeux de simulation de conflit armé, et les RTS, des jeux de stratégie.

# 2 Revue des notions nécessaires

## 2.1 Survol du sujet

Nous travaillerons ici à utiliser les modèles de Markov cachés pour permettre aux intelligences artificielles dans les jeux vidéo d'estimer de manière plus réaliste, humaine et efficace la position de son adversaire. Pour s'assurer que les intelligences artificielles soient de coriaces adversaires, ils doivent souvent tricher en obtenant de l'information qu'un utilisateur n'aurait pas ou en leur donnant certaines capacités surhumaines. Ce que nous voulons c'est de rendre le *bot* performant dans son jeu, mais ce avec les mêmes outils qu'un joueur humain posède.

## 2.2 Types de jeux sujets à la recherche

Parmi le peu d'articles qui traitent le sujet, deux sont très interessants. Chaque article porte sur un type de jeu distinct. Définissons brièvement ces types de jeux, les raisons pourquoi un travail sur les intelligences artificielles est nécessaire et comment une chaîne de Markov cachée peut résoudre les problèmes énoncés.

### 2.2.1 First Person Shooter

Le premier article, celui de Hladky et Bulitko [HLA08], vers lequel mes recherches m'ont menées traite les FPS's ( First Person Shooter ). Il s'agit d'un jeu de simulation de conflit armé dans lequel deux équipes, qui peuvent être de toutes tailles, s'affrontent et l'emporte lorsque tous les membres de l'équipe adverse sont éliminés. Imaginons un jeu où les équipes sont de taille 1. Ici, un joueur souhaite améliorer ses performances en compétitionnant contre un bot. Il devra donc trou-

ver, puis liquider son adversaire avant que ce dernier ne fasse de même. Ce jeu comprend trois dimensions et se joue à la première personne, cela signifie que le joueur voit ce qu'un être humain placé à cette même position verrait. Dans l'objectif de compenser pour leur lenteur, ou parfois même leur incapacité à déterminer la position de l'utilisateur, l'intelligence artifielle triche souvent dans ce type de jeu. Cette dernière possède parfois la position exacte du joueur quand ce dernier se trouve à une certaine distance, ou parfois, l'I.A. possède une rapidité et une précision inhumaine lui permettant de faire feu rapidement sur son adversaire dès qu'il se trouve dans son champ de vision. Cette situation peut être frustrante pour le joueur. Nous voulons donc programmer notre *bot* pour être intelligent, mais réaliste.

C'est ici que les modèles de semi-Markov cachés nous servirons. Nous expliquerons précisément ce qu'est un modèle de semi-Markov sous peu. Pour l'instant, comprenons comment un modèle de Markov caché peut nous aider à réaliser ce type d'intelligence artificielle. Les auteurs désirent utiliser les HMM ( Hidden Markov Models) dans le but d'estimer la position des joueurs sur une carte géographique. Nous verrons ici chaque position sur la carte comme étant nos états. La position de notre adversaire au temps $t$ est en fait l'état dans lequel se situe notre chaîne de Markov au temps $t$. Cette chaîne est cachée puisque nous ne connaissons pas précisément la position de notre adversaire mais nous avons néanmoins des observations. L'observation au temps $t$ est un vecteur contenant chacune des positions que nous voyons en tant que bot au temps $t$. Pour chacune des ces positions, nous savons si l'adversaire y est ou non. Le modèle se complique mais reste efficace en considérant que nous avons plusieurs adversaires et que ceux-ci ne sont pas distinguables.

### 2.2.2 Real-Time Strategy

Un autre type de jeu pour lequel certains chercheurs ont tenté d'utiliser les modèles de Markov caché est le RTS ( Real-Time Strategy) [SOU07]. Ce jeu se joue d'une vue aérienne en deux dimensions. Il s'agit d'un jeu de simulation de guerre où le joueur ne contrôle pas seulement un personnage, mais bien une base de commande ainsi qu'une armée. Alors que dans un FPS, les réflexes et la précision sont de mise, c'est la stratégie économique et la stratégie militaire qui sont mises de l'avant dans les RTS. Encore une fois, les adversaires controlés par une intelligence artificielle dans ce type de jeu ont souvent un comportement douteux. Parfois, leur base de commande recoivent un plus grand nombre de ressources et parfois même ils connaissent la position exacte des armées adverses en déplacement, leur permettant de tendre des ambuscades beaucoup plus planifiées que ce qu'un vrai joueur aurait pu faire. Encore une fois, un modèle de Markov caché sera utilisé ici pour permettre au *bot* d'estimer la position de ses adversaire sur une carte en deux dimensions.

Comme pour les FPS, les états de notre modèle markovien seront la totalité des positions sur la carte. Nous cherchons a identifié la position de troupes ennemes en déplacement. Encore une fois, nous avons ici plusieurs unités ennemies dont nous voulons estimer la position et certaines unités sont indistinguables. Nos observations sont encore une fois les positions que nous voyons en tant qu'intelligence artificielle. Nous verrons le modèle plus en détail dans la section suivante. Approfondissons désormais nos connaissances sur les divers processus de Markov dont il sera question.

## 2.3 Modèles de Markov à l'étude

### 2.3.1 Processus semi-Markovien (Semi-Markov Processes)

On trouve dans Ross [ROS10] un chapitre portant sur les processus semi-Markoviens. Nous pouvons imaginer une chaîne de Markov bien simple avec trois états par exemple. Dans le cas d'un processus Markovien classique, au temps $t$, nous serions à l'un des trois états de notre chaîne,$i$ par exemple, et au temps $t+1$, précisément une transition aurait eu lieu et nous serions maintenant à l'état $j$ avec probabilité $P_{ij}$. Dans un modèle semi-Markovien, nous restons un nombre de temps aléatoire à chaque état entre nos transitions. Nous restons donc à l'état $i$ du temps $t$ au temps $t+d$, où $d$ est une variable aléatoire, puis nous exécutons une transition à l'aide des probabilités de notre chaîne comme nous l'aurions fait dans le cas d'un modèle de Markov classique. Il est important de noter que ce laps de temps aléatoire entre les transitions peut être discret ou continu. Si le temps d'attente entre les transitions est constant de valeur 1, nous sommes face a une chaîne de Markov standard. Quelques propriétés peuvent s'en dégager.

Soit un modèle de semi-Markov à trois états. Nous restons un nombre de temps aléatoire à chaque état $i$ selon une variable aléatoire de moyenne $\mu_i$. Supposons que nous visitons tous les états l'un à la suite dse autres, en boucle. Soit $p_i$, La proportion de temps passé à l'état $i$, nous pouvons observer que :

$$p_i = \frac{\mu_i}{\mu_1 + \mu_2 + \mu_3} \tag{1}$$

Comment utiliser cette équation quelque peu évidente dans un cas où nous

ne visitons pas les états les uns à la suite des autres, mais plutôt dans un cas où nous passons d'un état à un autre avec une certaine probabilité. Considérons donc que nos trois états suivent une chaîne de Markov, plusieurs résultats existent déja quant à la proportion de temps à long terme passé dans un certain état. Dans le théorie des processus de Markov, nous avons défini $\pi_i$ comme étant la proportion des transitions allant vers l'était $i$. Comme nous savons que nous resterons à l'état $i$ en moyenne $\mu_i$ unité de temps, il est évident qu'il faut utiliser $\pi_i$ pour pondérer la proportion de temps passé dans un état. Il serait donc logique d'en déduire que, ici :

$$p_i = \frac{\pi_i \mu_i}{\pi_1 \mu_1 + \pi_2 \mu_2 + \pi_3 \mu_3} \tag{2}$$

Donc, de manière générale, si nous sommes face à un processus de semi-Markov, avec $n$ états :

$$p_i = \frac{\pi_i \mu_i}{\sum_{i=1}^{n} \pi_i \mu_i} \tag{3}$$

Bien que ce résultat ne soit pas directement utile, il est intéressant de voir qu'il peut être parfois simple d'utiliser ce que nous savons des processus Markovien standards pour gérer les modèles de semi-Markov.

### 2.3.2 Modèles de Markov cachés

Définissons rapidement ce que nous aurons de besoin sur les modèles de Markov cachés à l'aide des informations obtenues dans le cours MAT998B dispensé à l'Université du Québec à Montréal. Un modèle de Markov cachée peut être décrit un peu rapidement comme étant une chaîne de Markov standard dont les états sont cachés. C'est-à-dire que nous avons un processsus Markovien dont les probabilités de transition ne dépendent que de l'état dans lequel nous nous trouvons. Cependant, les données que nous observons ne sont pas les états, mais bien une variable aléatoire sur les états. Au temps $t$, notre chaîne est à l'état $i$, mais nous n'obtenons pas l'état $i$ comme observation, mais plutot le résultat d'une variable aléatoire qui dépend de l'état dans lequel nous nous trouvons. Lorsque nous travaillons avec des modèles de Markov cachés, le défi est justement de déterminer quels sont les états de la chaîne de Markov et quelles sont les probabilités de transitions en n'ayant que nos observations comme données.

Plusieurs variables entre en compte, définissons donc quelques notations. Comme pour un processus de Markov standard, nous avons tout d'abord un vecteur d'état, par exemple $S = \{s_1, s_2, ..., s_n\}$, ainsi qu'un vecteur de probabilité initial $\mu = \{\mu_1, \mu_2, ..., \mu_n\}$, ce dernier nous informe de la probabilité de débuter notre processus markovien à chacun de nos $n$ états. Nous aurons bien entendu une matrice de taille $n$ par $n$ contenant les probabilités $a_{ij}$ de transition de l'état $i$ vers l'état $j$. Nous allons dénoter $Q = q_1, q_2....q_T$ la séquence des états visités lors des $T$ transitions. Par contre, nous travaillons ici dans un modèle caché, ce que nous aurons comme information, est une variable aléatoire associée à chacun des états, notons donc $O = o_1, o_2, ..., o_T$ la séquence d'observations obtenue de nos $T$ états

visités. Finalement, comme nous travaillons dans un modèle caché, nous devons avoir $b_i(j)$ qui est défini comme $P[o_t = j|q_t = s_i]$ .

Plusieurs algorithmes ont été mis au point dans le but de résoudre certaines de ces questions. L'algorithme *foward-backward*, qui sera détaillé dans la prochaine section, nous permet entre autre d'estimer la probabilité d'être à un certain état au moment $t$ ayant obtenu les $t$ observations précédentes. Dans un objectif d'application aux jeux vidéo, ce sera cet algorithme qui nous intéressera principalement. Décrivons cet algorithme brièvement. Définissons donc $\alpha_t(i) = P[o_1, o_2, ...o_t$ et $q_t = Si]$. Nous allons calculer cette valeur de manière récusive. Calculons donc la première valeur possible :

$\alpha_1(i) = P[o_1, q_t = Si] = P(q_1 = S_i)P(o_1|q_1 = S_i) = \mu_i b_i(o_1)$

Calculons donc $\alpha_{t+1}(j)$ :

$$
\begin{aligned}
\alpha_{t+1}(j) &= P(o_1, o_2, ...o_t, o_{t+1} \text{ et } q_{t+1} = S_j) \\
&= \sum_i^N P(o_1, o_2, ...o_t, o_{t+1} \text{ et } q_t = S_i \text{ et } q_{t+1} = S_j) \\
&= \sum_i^N P(o_{t+1} \text{ et } q_{t+1} = S_j | o_1, o_2, ...o_t \text{ et } q_t = S_i) P(o_1, o_2, ...o_t \text{ et } q_t = S_i) \\
&= \sum_i^N P(o_{t+1} \text{ et } q_{t+1} = S_j | q_t = S_i) \alpha_t(i) \\
&= \sum_i^N P(o_{t+1} | q_{t+1} = S_j) P(q_{t+1} = S_j | q_t = S_i) \alpha_t(i) \\
&= \sum_i^N b_j(o_{t+1}) a_{ij} \alpha_t(i) \\
&= b_j(o_{t+1}) \sum_i^N a_{ij} \alpha_t(i)
\end{aligned}
\tag{4}
$$

Grâce aux $\alpha$ , nous pourrons calculer :

$P(o_1, o_2, ...o_t) = \sum_{i=1}^N \alpha_t(i)$

Ainsi que :

$P(q_t = S_i | o_1, o_2, ...o_t) = \frac{\alpha_t(i)}{\sum_{i=1}^N \alpha_t(i)}$

Il y existe évidemment d'autres techniques dans l'objectif de répondre à d'autres problématiques. Entre autre, il est possible d'obtenir la série d'états visités jusqu'au temps $t$ qui maximise la probabilité d'avoir obtenu nos observations grâce

au principe de Bellman. De plus, l'algorithme de *Baume-Welch* nous donne la chance d'approximer les probabilités de transition de la chaîne de Markov sous-jacente en utilisant nos observations. Par contre, ces outils ne seront utilisés ici.

# 3 Modèles de semi-Markov cachés (HSMM)

## 3.1 Dévelopement du modèle

Nous pouvons donc désormais aborder le sujet des modèles de semi-Markov cachés. Souvenons-nous que dans un modèle de semi-Markov, nous resterons un nombre de temps aléatoire dans un certain état. Nous travaillerons ici dans le cas où cette variable aléatoire est dicrète. Un modèle de semi-Markov caché possède la même structure qu'un modèle de Markov caché sauf que la chaîne cachée est semi-Markovienne. Nous avons donc une chaîne de Markov bien standard où nous avons des états et des probabilités de transitions. Nous effecturons une transition de l'état $q$ vers l'état $q'$ avec probabilité $P_{qq'}$, puis, nous resterons à l'état $q'$ un nombre de temps aléatoire discret, disons $l$ unités de temps. Comme la chaîne ici est cachée, nous ne voyons pas l'état dans lequel nous nous trouvons mais plutôt une variable aléatoire en lien avec cet état. Comme nous resterons $l$ unités de temps à l'état $q'$, nous obtiendrons $l$ observations de la variable aléatoire associée à l'état $q'$. Chaque état va donc émettre une série d'observations, et non une seule.

Définissons certaines notations. Ces dernières proviennent de l'article de Murphy [MUR02] qui fut ma principale source sur le sujet. Nous dénoterons $Y(G_t)$ la séquence d'observations émise par $G_t$. Ce dernier est défini comme étant " l'état généralisé " dans Murphy, c'est-à-dire qu'il contient $Q_t$, l'état de notre chaîne de Markov au temps $t$ ainsi que $L_t$, le temps que la chaîne de semi-Markov restera à l'état dans lequel nous sommes au temps $t$. $Y(G_t^+)$ représentera toutes les observations qui suivront $Y(G_t)$ et $Y(G_t^-)$ représentera donc la totalié des observations qui ont précédé $Y(G_t)$. De plus, définissons $G_{t_n}$ comme étant l'état suivant $G_t$ et $G_{t_p}$ l'état précédent. Finalement, désignons $O_t(q, l)$ comme

étant l'équivalent de $b_i(j)$ pour une chaîne semi-Markovienne, c'est-à-dire que : $O_t(q,l) = P(Y(G_t)|Q_t = q, L_t = l)$. Voyons désormais comment le fait que nous travaillons désormais avec une chaîne de Semi-Markov affecte l'algorithme *foward*. Notons que plusieurs notations ont été introduites ici, et que d'un article à l'autre, les notations varient. Je tenterai d'être le plus clair possible lors de changement de notation, mais néanmoins il sera sûrement nécessaire de venir se référer plusieurs fois au paragraphe ci-dessus.

## 3.2 Modifications de l'algorithme *foward-backward*

Souvenons que pour un modèle de Markov caché, nous avions défini :

$\alpha_t(i) = P(o_1, o_2, ...o_t \text{ et } q_t = Si)$.

Pour un modèle de semi-Markov caché, nous devrons remplacer $S_i$, par l'état généralisé $g$. De plus, la totalité des observations précédentes sera maintenant $Y(G_t)$ ainsi que $Y(G_t^-)$. Nous obtiendrons donc que :

$\alpha_t(g) = P(Y(G_t^-), Y(G_t) \text{ et } G_t = g)$

Vérifions comment obtenir la forme récursive de nos $\alpha$.

$$
\begin{aligned}
\alpha_t(g) &= P(Y(G_t^-), Y(G_t), G_t = g) \\
&= \sum_{g'} P(Y(G_t^-), Y(G_t), G_t = g, G_{t_p} = g') \\
&= \sum_{g'} P(Y(G_t)|G_t = g, G_{t_p} = g', Y(G_t^-))P(G_t = g, G_{t_p} = g', Y(G_t^-)) \\
&= \sum_{g'} P(Y(G_t)|G_t = g)P(G_t = g|G_{t_p} = g', Y(G_t^-))P(G_{t_p} = g', Y(G_t^-)) \\
&= \sum_{g'} P(Y(G_t)|G_t = g)P(G_t = g|G_{t_p} = g')P(G_{t_p} = g', Y(G_t^-)) \\
&= O_t(g) \sum_{g'} P(g|g')\alpha_{t_p}(g')
\end{aligned}
\tag{5}
$$

Souvenons nous que $O_t(g)$ est en quelque sorte l'équivalent de $b_i(j)$ et que $P(g|g') = a_{g'g}$, la probabilité de transition de l'état $g'$ vers l'état $g$. Nous avons donc la même formule que nous avions dans le cas d'un modèle de Markov caché classique. Néanmoins, le fait de travailler avec $G_t$ peut parfois être contraignant puisqu'il s'agit d'un couple de variables. Traduisons la formule ci-haut de manière à y observer $Q_t$, l'état de notre chaîne, et $L_t$, la longueur du segment de temps que nous avons demeuré dans l'état. Murphy introduit finalement une dernière variable dans son modèle qui ne se retrouve pas dans d'autres papiers portant sur les modèles de semi-Markov cachés. Il définit alors $F_t$ comme étant un variable booléenne dans l'objectif de connaître où en somme nous par rapport à $l$, le nombre de temps que nous avons passé dans notre état. Nous posons $F_t = 1$ si nous sommes entre deux segments de temps, de manière général si $Q_t \neq Q_{t+1}$, à l'exception

bien sur du cas ou nous quittons $Q_t$ pour y revenir. $F_t = 0$ alors si nous resterons au même états au temps $t + 1$ que nous étions au temps $t$. En gardant cela en tête, définissons un nouveau $\alpha$.

Définissons : $\alpha_t(q, l) = P(Q_t = q, L_t = l, F_t = 1, y_{1:t})$

Nous travaillons donc ici au dernier pas avant un changement d'état, puisque $F_t = 1$. De plus, souvenons-nous que $y_{1:t}$ représente les $t$ premières observations, précédemment identifer comme $Y(G_t), Y(G_{t_p})$. Nous pouvons donc utiliser le $\alpha_t(g)$ (équations (5)) que nous avons calculé précédemment, en se souvenant que nous venons d'écouler nos $l$ unités de temps à l'état $q$ .

$$\begin{aligned}\alpha_t(q, l) &= P(Q_t = q, L_t = l, F_t = 1, y_{1:t}) \\ &= P(y_{t-l+1:t}|q, l) \sum_{q'} \sum_{l'} P(q, l|q', l')\alpha_{t_p}(q', l') \end{aligned} \tag{6}$$

Par la suite, une supposition logique est faite dans le papier de Murphy pour simplifier l'algorithme *foward*. Il est supposé que $P(q, l|q', l') = P(q|q')P(l|q)$. Cette suppostion peut se justifié aisément. Nous supposons ici que la probabilité de se déplacer vers un nouvel état $q$ et d'y rester un nombre de temps $l$ sachant où nous sommes actuellement $q'$ et combien de temps nous y avons demeurer $l'$ peut se traduire plus simplement. C'est en fait la probabilité de passer de l'état $q'$ vers l'état $q$ et la probabilité d'y rester $l$ unité de temps sachant que nous sommes désormais à l'état $q$. Ceci suppose en fait que la probabilité de transition vers un état ne dépend en fait que de l'état dans lequel nous nous trouvons, et non du nombre de temps que nous y avons passé, et cela suppose également que le nombre

de temps que nous resterons à un état ne dépend que de cet état, et non du passé. Ces suggestions sont justifiables dans la plupart des applications et s'apprêtent relativement bien à l'application que nous en ferons plus tard dans ce rapport. Voici maintenant comment cette supposition affecte $\alpha_t(q,l)$ :

$$\begin{aligned}
\alpha_t(q,l) &= P(y_{t-l+1:t}|q,l)\sum_{q'}\sum_{l'} P(q,l|q',l')\alpha_{t_p}(q',l') \\
&= P(y_{t-l+1:t}|q,l)\sum_{q'}\sum_{l'} P(q|q')P(l|q)\alpha_{t_p}(q',l') \\
&= P(y_{t-l+1:t}|q,l)P(l|q)\sum_{q'} P(q|q')\left(\sum_{l'}\alpha_{t_p}(q',l')\right)
\end{aligned} \tag{7}$$

Finalement, nous introduirons une nouvelle variable $\alpha$, cette dernière sera la version finale, et celle que nous utiliserons pour nos applications futures. Dans un objectif d'épurer le tout et d'obtenir un $\alpha$ pour faire de l'inférence uniquement sur les états de la chaîne de markov, nous allons définir :

$$\begin{aligned}
\alpha_t(q) &= P(Q_t = q, F_t = 1, y_{1:t}) \\
&= \sum_l \alpha_t(q,l) \\
&= \sum_l P(y_{t-l+1:t}|q,l)P(l|q)\sum_{q'} P(q|q')\left(\sum_{l'}\alpha_{t_p}(q',l')\right) \\
&= \sum_l P(y_{t-l+1:t}|q,l)P(l|q)\sum_{q'} P(q|q')\alpha_{t_p}(q')
\end{aligned} \tag{8}$$

Notons finalement que l'article de Murphy aborde plusieurs autres sujets, mais que seul cet algorithme nous importe en ce qui concerne notre application. Nous allons donc désormais aborder le sujet de la programmation d'intelligence artificielle plus performante grâce aux modèles de semi-Markov caché.

# 4 Utilisation de ce modèle à certains jeux vidéos

Il sera ici question de l'amélioration potentielle de la capacité d'une intelligence artificielle à estimer la position géographique de ses adversaires en utilisant les processus semi-Markoviens cachés. Le *bot* devra déterminer la position de son opposant en ne disposant que de quelques observations. Souvenous-nous, que nous pouvions obtenir les probabilités d'être à divers états en utilisant nos observations grâce à l'algorithme *foward*. Nous avions mentionné l'égalité suivante à la section 2.4.1 : $P(q_t = S_i|o_1, o_2, ...o_t) = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)}$

## 4.1 First Person Shooter

L'article dont il sera question ici est le papier de Hladky et Bulitko [HLA08]. Celui-ci aborde deux modèles qui pourraient être utiles à la prédiction de la position d'adversaires dans un FPS. L'un de ces modèles étant bien entendu les processus de Semi-Markov cachés. L'autre méthode à l'essai ici est le filtre particulaire ( Particle Filters ). Nous allons brièvement expliquer ce qu'est le filtre particulaire puis nous irons au vif du sujet, l'utilisation des modèles de semi-Markov cachés. Nous terminerons avec une explication des techniques qui furent utilisés pour déterminer et obtenir les informations nécessaires à l'utilisation de ce modèle ainsi qu'une courte explication des techniques expérimentales.

### 4.1.1 Filtres particulaires

Tout d'abord, il est important de comprendre que mon rapport porte sur les modèles de semi-Markov cachés. Comprendre parfaitement la théorie des filtre particulaires est une lourde tâche qui, quoique très intéressante, n'est pas la di-

rection que j'ai décidé de suivre. L'introduction fait ici sera très peu informative, mais suffisant pour voir le potentiel de cette méthode. Les filtres particulaires, aussi appelés méthodes de Monte-Carlo séquentielles, sont des techniques d'estimations basées principalement sur la simulation, ayant pour objectif d'estimer une densité à postériori grâce aux méthodes Bayésiennes . Cette méthode est surtout utilisée pour les modèles de Markov cachés dans l'objectif d'estimer de l'information sur notre modèle caché, à partir de nos observations. La technique employée par les auteurs de l'article utilise l'échantillonnage avec rééchantillonnage par importance ( Sampling Importance Resampling ). Cette technique approxime la probabilité à postériori, ici la probabilité d'être à un état sachant nos observations, par des particules pondérées. Un rapport entier pourrait être fait dans l'objectif d'expliquer cette technique, et de nombreux articles ont déjà été faits sur le sujet. Il pourrait être intéressant comme piste d'étudier davantage ce que nous offre les filtres particulaires pour analyser les modèles de Markov cachés. Néanmoins, l'article duquel je m'inspire obtient comme conclusions que les modèles de semi-Markov cachés, que nous avons commencé à élaborer, sont plus performants. Continuons donc dans cette direction.

### 4.1.2 Utilisations des modèles de semi-Markov cachés

Nous allons ici décrire comment utiliser les modèles de semi-Markov cachés pour aider les intelligences artificielles à estimer la position de ses adversaires. Définissons d'abord nos variables. Le jeu examiné lors de ce papier est *Counter-Strike*. Ce jeu est l'un des précurseurs des *FPS*, il s'adapte bien à notre expérimentation car, il est encore joué aujourd'hui et il serait facile d'adapter le travail fait sur ce jeu à d'autres *First-person shooter*. Dans ce jeu, deux équipes s'affrontent, les

méchants *Terrorists* et les gentils *Counter-Terrorist*. L'objectif des méchants est de poser une bombe à l'un des deux sites possibles, les gentils doivent empêcher la chose. Si la bombe explose avant un certain délai de temps, les terroristes l'emportent, sinon ils perdent. En tout temps, si une équipe élimine tous les membres de l'équipe adverse elle est victorieuse. Plusieurs cartes sont disponibles comme lieux de combat. Supposons qu'un utilisateur commence une partie, un joueur contre un joueur, et il affronte ici un *bot*.

Nous allons prendre une carte, puis la quadriller de sorte à ce que chaque case soit de la taille d'un joueur. Puis, nous allons considérer ces cases, les diverses positions possibles des joueurs sur la carte comme étant les états de notre modèle de Markov. Nous connaissons donc ici la totalité de nos états, ainsi que la possiblité de transition des états entre eux. Rappelons-nous que nous sommes ici l'intelligence artificielle qui compétitionne avec le joueur. La chaîne est à l'état $i$ au temps $t$ si le joueur, notre adversaire, est dans à la position $i$ au temps $t$. Nous sommes en présence d'un modèle de Markov caché puisqu'au temps $t$, nous ne pouvons pas savoir que notre adversaire se trouve à la case $i$. Nous avons néanmoins des observations pour nous aider. Ici, une observation est un vecteur de couple. Chaque couple est composé d'une case de notre champ de vision, ainsi que d'un booléen indiquant si nous y voyons l'adversaire ou non. Une observation renferme donc la totalité des positions que nous voyons, en précisant si l'adversaire se trouve dans l'une de ses cases ou non. En tant qu'inteligence artificielle, nous cherchons à estimer la position la plus probable de notre adversaire en sachant nos observations. Nous devrons donc calculer la probabilité qu'il se trouvent à chacun des états. Nous savons qu'à l'aide d'un modèle de semi-Markov caché nous pourrons calculer : $P(q_t = S_i | o_1, o_2, ...o_t)$.

Il nous manque néanmoins encore quelques informations afin de pouvoir utiliser un modèle de semi-Markov caché. $\mu$, la distribution initiale est facile à obtenir. C'est parce que ce jeu possède déja ce vecteur $\mu$. Pour chacune des cartes disponibles, il y a déja des positions de départ attitrées pour chacune des équipes. Dans cette ensemble de cases disponibles pour chacune des équipes, la sélection se fait de manière uniforme. Comme *Counter-Strike* permet un maximum de 12 joueurs par équipes, il y a donc 12 possibilités de positions initiales pour notre adversaires, et celles-ci sont proches les unes des autres, mais sont surtout connues par la totalité des joueurs. Ce n'est donc pas tricher que de possèder cette information en tant qu'intelligence artificielle. Pour utiliser ce que nous savions sur les modèles de semi-Markov caché, nous devons travailler dans un contexte discret. Nous allons discretiser le jeux en actualisant notre modèle à tous les 0.45 secondes. Entre $t$ et $t+1$ il se sera donc écouler 0.45 secondes, et c'est à ce moment que l'A.I. regardera à nouveau les position qu'il observe et tentera de deviner où se trouve son adversaire. Maintenant, expliquons d'où vient ce 0.45 secondes et pourquoi cette valeur peut nous aider. C'est parce que les autres informations nécessaires seront obtenues de manière empirique à partir de *game logs*. Ces journaux de bord sont en quelque sorte une collection de photos prises à un instant du match contenant toutes les informations de ce match. On y trouve le nombre de points de vie de tous les joueurs, ce qu'ils voient, mais bien entendu, ce qui compte le plus pour nous, leurs positions. Ces *game logs* contiennent une image à chaque tranche de 0.45 secondes de jeu. Les auteurs de l'article se sont donc servi d'un échantillon substantiel de *game logs* pour estimer les informations nécessaires à l'utilisation d'un modèle de semi-Markov caché. La matrice des probabilités de transition fut donc estimée de manière empirique en utilisant les journaux de bord. De même

manière, la fonction de durée fut estimée elle aussi empiriquement.Rappellons qu'il s'agit de la fonction aléatoire qui détermine combien de temps un joueur reste à un certain état. Lorsque un joueur était au même état au temps $t$ et au temps $t+1$, une variable, $c$ par exemple, augmente de 1. Nous pourrons donc estimer le nombre de temps moyen passé à une certaine position.

Finalement, il ne nous reste que la fonction d'observation à construire. Nous devons d'abord analyser chaque position visible de la part de notre intelligence articifielle. Puis vérifier si notre adversaire s'y trouve. Ce qui complique la tâche c'est que nous devons considérer que ce que nous voyons est en trois dimensions et que parfois, des objets, une boîte par exemple, nous fera suffisament d'obstruction pour nous empêcher de nous assurer que le joueur adverse n'est pas caché derrière. Il faudra donc à la fin utiliser l'infomation en trois dimensions que nous détenons pour conclure où l'adversaire ne se trouve pas dans notre liste d'état, le quadrillage deux dimensions de notre carte. Plus précisément, ils définissent : $P(O_t|s_t = g) = \frac{|W_t \cap X(g)}{|X(g)|}$ où $W_t$ est l'ensemble des cubes visibles en trois dimensions et $X(g)$ l'ensemble des cubes centré sur la case $g$. Notons qu'il faudra mettre à jour certaines probabilités après avoir effectué nos observations. Si nous observons un adversaire à la position $g$, nous allons explicitement fixer $P(s_t = g|O_{1:t}) = 1$, et pour toutes les positions $g'$ où nous savons qu'aucun adversaire est présent, nous allons fixer $P(s_t = g'|O_{1:t}) = 0$.

Nous pouvons finalement utiliser l'algorithme *foward* afin de calculer nos $\alpha$. Voici la version de l'algorithme *foward* utilisé par les auteurs. *Notons ici que ces derniers semblent avoir confondu $\alpha_t(i)$ avec $P(s_t|O_{1:t})$. Voici donc la version corrigé :*

$$\begin{aligned}\alpha_t(i) &= \sum_d P(O_{t-d+1:t}|s_t)P(d|s_t)\sum_{s_{t-d}} P(s_t|s_{t-d})\alpha_{t-d}(s_{t-d}) \\ &= \sum_d P(d|s_t)\left(\prod_{u=t-d+1}^{t} P(O_u|s_t)\right)\sum_{s_{t-d}} P(s_t|s_{t-d})\alpha_{t-d}(s_{t-d})\end{aligned} \tag{9}$$

Notons quelques observations finalement. Tout d'abord, les notations ont encore changés. Ici, nous utilisons $d$ pour identifier la durée de temps et $s_t$ pour l'état auquel notre adversaire se trouve au temps $t$. Il est aussi intéressant de constater que les auteurs font la suppostion que $P(O_{t-d+1:t}|s_t) = \prod_{u=t-d+1}^{t} P(O_u|s_t)$. Ils supposent donc que chaque observation est indépendante les unes des autres et qu'une observation ne dépend que de l'état dans lequel se trouve le ou les adversaires. Bien que cette supposition ne soit pas facile à accepter, elle simplifie beaucoup les calculs et n'affecte probablement que légèrement la qualité de notre estimation. Bien sûr, comme nous avons vu précédemment, nous pourrions ici utiliser : $P(q_t = S_i|o_1, o_2, ...o_t) = \frac{\alpha_t(i)}{\sum_{i=1}^{N}\alpha_t(i)}$ pour estimer la probabilité de chacune des positions. Notons finalment que nous pourrions facilement appliquer ce modèle à des tailles d'équipes plus grandes, les expérimentations ont d'ailleurs été faites lors de parties où 5 joueurs affrontaient 5 intelligences artificielles.

### 4.1.3 Expérience et résultats

Les auteurs ont finalement mis leurs différents modèles à l'épreuve. L'objectif était d'avoir un I.A qui évalue bien la position de son adversaire, et qui fait des erreurs typiquement humaines. Ils ont alors évalué la perfomance de leur *bot* de deux manières, l'une considérait la différence entre la véritable position de

l'adversaire et la prédiction du *bot*. Pour évaluer si l'erreur est humaine, les expérimenteurs ont fait observer des parties à certains joueurs professionnels et ont régulièrement demandé à ces joueurs d'indiquer sur la carte où étaient les joueurs adverses selon eux. Ces données ont permis aux auteurs de calculer la différence moyenne entre la prédiction de l'intelligence artificielle et celle d'un joueur humain. Ces deux statstiques furent calculées à partir d'un échantillon de 50 parties. Ils ont finalement intégré ces moyennes d'erreur sur des tableaux croisés et ont déterminé que les meilleurs prédicteurs étaient ceux non-dominés. Leurs résultats démontrent que les modèles de semi-Markov cachés performent mieux que les filtres particulaires. De plus, les prédictions du *bot* sont plus précises que celle d'un joueur expérimenté et sont beaucoup plus humaines que celle des intelligences artificielles actuellement utilisés.

Il serait néanmoins intéressant de voir quelles améliorations pourraient être faites. Comment considérer le son dans les observations par exemple ou comment adapter les prédictions lorsqu'un adversaire est près de la mort ? Les résultats sont néanmoins très intéressants. Voyons rapidement comment appliquer ce modèle pour un autre jeu.

## 4.2 Real-Time Strategy

Cet article de Southey, Loh & Wilkinson [SOU07] est une référence utilisée par celui dont nous venons de parler. Bien qu'il ne soit pas le coeur de ma recherche, je crois que d'en glisser un mot peut apporter quelque chose. Il traite de *RTS*, un jeu de simulation de guerre où l'utilisateur agit comme commandant d'une armée et ce, avec une vue aérienne. Pour combler un certain manque, nous

allons juger que les auteurs ont utilisé une méthodologie similaire à ce qui fut utilisée pour les *FPS*. Nous nous concentrerons principalement sur les différences que causent le changement de jeu. Il est important de considérer que le but de l'article ici n'était pas d'améliorer des intelligences artificielles mais plutot d'être capable de faire de l'inférence sur la trajectoire d'un agent avec seulement quelques observations partielles ainsi que le départ et l'arrivé. Ils se sont plutot servis des *RTS* comme d'une platforme pour faire leurs expérimentations.

### 4.2.1 Utilisations des modèles de Semi-Markov cachés

Les variables ici sont similaires à ce que nous avons vu précédemment. Souvenons-nous, que ce jeu ne comporte que deux dimensions. Il est donc beaucoup plus simple d'imaginer un quadrillage en deux dimensions de la carte ou chacun de ces carrés est donc une position possible d'un unité adversaire. Ces positions sont les états de notre chaîne de Markov et sont connus par l'ensemble des joueurs. Encore une fois, $\mu$ la distribution initiale est déjà implenté dans le jeu et connu des joueurs. Finalement il n'est pas explicitement décrit dans le texte comment son obtenue les autres paramètres nécéssaires à l'utilisation d'un modèle de semi-Markov caché. Nous pourrions supposé une méthodologie similaire qu'en ce qui concerne les *FPS*. Une des différences les plus importantes concerne les observations. Ici, le joueur, et donc le *bot* qui est supposé être un joueur, agit en tant que commandant d'une armée et sa vision lui est donnée par ses troupes. Ce qu'un joueur voit est tout ce qui se trouve a une certaine distance d'au moins une de ces unités. Ce que chacune des unités est en quelque sorte entourée d'un cercle de lumière représentant sa vision et c'est l'ensemble de tous les cercles que le joueur voit. Il est un peu plus simple de travailler avec les observations ici puisque nous

sommes directemment en deux dimensions, nous n'avons pas à faire ce transfert de trois vers deux dimensions. Notons finalement qu'une autre distinction importante est que les auteurs souhaitaient estimer le chemin emprunter d'un joueur n'ayant que quelques observations ainsi que le point de départ et d'arrivé. Pour alléger la rapidité d'exécution, ils ont diminué le nombre de *endpoints*, points de départ et d'arrivée, possibles, en procédant par abstraction sur le graphe.

Avec tous ces outils, les auteurs ont utilisés l'algorithme *foward* de la manière suivante :

$$\alpha_t(S_t) = \sum_d P(d|S_t) \left(\prod_{u=t-d+1}^{t} P(O_u|S_t)\right) \sum_{S_j} P(S_t|S_j)\alpha_{t-d}(S_j)$$

### 4.2.2 Expérience et résultats

L'expérience fut mené sur le célèbre jeu *Warcraft III*. Ils ont d'abord créé leurs propres cartes de jeu. Dans cette carte relativement simple, ilsl sont arrivés à une précision plus que satisfaisante pour eux. Leur prédiction d'un chemin dit embrouillé est extrêmement précise. Notons que ici les auteurs tentent de déterminer un chemin bruyant, ils ne se restreignent donc pas à une seule succession d'état, ils s'autorisent à considérer des chemins où les unités adverses titubent un brin. Sur des plus grandes cartes, le modèle fonctionne relativement bien, par contre le temps d'exécution est lourd, et l'estimation perd beaucoup de précisions lorsque plus d'abstractions sont faites sur le graphe. Les auteurs semblent néanmoins voir les résultats d'un bon oeil et il serait intéressant de voir comment se débrouillerait ce modèle pour un problème dans la vie réelle.

# 5 Adaptation potentielle vers un nouveau type de jeu

À titre d'ouverture, essayons de voir si ce que nous avons vu pourrait s'appliquer à d'autre jeu. Très fort probablement, à vrai dire, nous avons abordé le sujet des *RTS* dans l'unique but d'observer que ce modèle pouvait bien s'adapter à différents jeux. Nous verrons ici les grandes lignes d'une adaptation potentielle vers un autre jeu, *Leagues of Legends*. Nous verrons pourquoi il serait intéressant de vouloir utiliser notre modèle pour les intelligences artificielles dans ce jeu. Par la suite, nous verrons brièvement comment pourrions obtenir les informations nécessaires à l'utilisation de l'algorithme *foward*.

## 5.1 Justification

Ce papier porte les modèles de semi-Markov cachés, et non sur les jeux vidéo. Je vais donc tenter d'être le plus bref possible pour justifier en quoi ce modèle pourrait être intéressant ici. Tout d'abord, il faut comprendre qu'il s'agit du jeu compétitif le plus joué en ligne présentement. Plusieurs tournois majeurs ont régulièrement lieu. Il s'agit présentemment du jeu de l'heure dans le domaine des jeux compétitifs. Il serait donc intéressant pour les nouveaux joueurs , et même les habitués, de pouvoir s'entraîner en affrontant des *bots*. Malheureusement, ce n'est pas exactement possible. Expliquons rapidemment le fonctionnement du jeu. Dans ce dernier, deux équipes de cinq joueurs s'affrontent. Chaque équipe possède un château et le château des deux équipes est séparé par une jungle dense. Trois chemins traversent la jungle et relient les deux chateaux. Typiquement, un joueur de chaque équipe emprunte le chemin du haut, un joueur, le chemin du

millieu et deux joueurs le chemin du bas. Comparativement à un FPS où les affrontements sont courts une fois qu'on est en contact avec un adversaire, ici des joueurs opposés s'affrontent constamment dans des batailles plutôt longues. De plus, ce jeu utilise un moteur similaire à celui que l'on retrouve dans les *RTS* et se joue d'une vue aérienne. Le cinquième joueur rode dans la jungle et attend une bonne opportunité pour tendre une embuscade à un adversaire dans l'un des chemins et donner un surnombre à son équipe dans cette voie pour un certain temps. Ce rôle, appelé *jungler*, étant un peu plus complexe, ne peut pas être rempli par une intelligence artificielle. Quand une équipe de cinq débutent une partie contre cinq *bots*, ceux-ci n'ont pas de *jungler* et envoie tout simplement leur cinquième joueur se battre sur le chemin au nord. Il est donc impossible pour de vrais joueurs de réellement se pratiquer contre des intelligences artificielles, puisque toutes les vraies équipes possèderont un *jungler*. Ce rôle ne peut pas être rempli par une *bot* pour l'instant car il est supposément un brin trop complexe à programmer. Pour savoir quand intervenir dans un des trois chemins et avoir un réel impact, il faut être expérimenté, bien connaître nos adversaires mais surtout, savoir la position des autres joueurs, principalement celle du *jungler* adverse. Pour que le *jungler* ait un effet bénéfique dans une ligne, il ne faut pas que le *jungler* adverse soit lui aussi près de ce chemin. Le défi pour un *jungler*, est donc souvent d'être capable de bien estimer la position du *jungler* adverse.

## 5.2 Utilisations des modèles de Semi-Markov cachés

La carte dans laquelle les joueurs s'affrontent est ici en deux dimensions, nous allons encore une fois utiliser un quadrillage de cette carte et voir chacune des cases comme une position possible de nos adversaires. Ces positions formeront

les états de notre chaîne. Ici, la carte est connue de tous les joueurs, la totalité des états sont donc connus par tous. Encore une fois, le vecteur de probabilité initiale est directement inclus dans le jeu. Ici, il n'y a que 5 positions de départ possible, chacun des joueurs est débutent donc à l'une de ses positions. Il est possible d'imaginer que nous pourrions estimer les probabilités de transition ainsi que la fonction du durée de manière à l'aide de moyennes empiriques comme il avait été fait pour les *FPS*. La particularité ici provient encore une fois des observations. Chaque joueur voit ses coéquipiers ainsi que tout ce qui entourent ceux-ci à une certaine distance. Dans un partie traditionnelle les joueurs sont souvent capables de voir la totalité de leurs adversaires à l'exception du *jungler* adverse, qui rode dans la jungle. Nos intelligences artificielles auraient la totalité de ce qu'ils voient comme information. Mais en plus de la position du joueur, il sera important qu'il note les points de vie de ses adversaires. Quand un ennemi est dans notre champ de vision, il est possible de savoir le nombre de points de vie qu'il lui reste. Les points de vie sont une manière d'évaluer l'état de santé d'un joueur, lorsqu'un joueur à 0 point de vie, il meurt. Les points de vie sont partie intégrante de la prise de décision de tous joueurs, mais surtout du *jungler*. Il voudra tendre des embuscades sur des adversaires faibles afin de pouvoir les éliminer facilement. Finalement, le temps est un facteur important dans ce jeu, certaines périodes charnières sont plus opportunes pour des attaques. Toutes ces informations pourraient faire parties d'un vecteur d'observations que possèderaient les intelligences artificielles. En utilisant les informations décrites ci-haut, nous pourrions utiliser l'algorithme *foward* comme nous l'avons décrit dans les chapitres précédents et espérer ainsi pouvoir améliorer les *bots* dans *League of Legends*.

# 6 Conclusion

Nous avons tout d'abord vu qu'est-ce qu'un modèle de semi-Markov caché. Comment se transcrit l'algorithme *foward* dans cette situation. Nous avons par la suite vu comment poser nos variables pour utiliser les modèles de semi-Markov cachés dans l'objectif de permettre aux intelligences artificilles dans les jeux vidéo à mieux prévoir la position possible de ces adversaires. Nous avons finalement vu comment utiliser les modèles de semi-Markov cachés dans un jeu comme *Counter-Strike*.

Pour terminer, dans ce type de jeu, il serait intéressant d'en considérer plus dans les observations que simplement ce que voit l'intelligence artificielle. Le temps, les points de vie des joueurs et surtout le son, sont des facteurs que considère les joueurs humains et que doivent considérer les *bots* pour bien les imiter. Il serait lourd et difficile d'inclure tout ces informations dans un modèle comme celui-ci, mais rendrait les parties contre des *bots* tellement plus compétitives.

Finalement, il serait intéressant de creuser plus profondémment dans les modèles de semi-Markov cachés, les filtres particulaires et les autres modèles du genre pour voir s'il n'y aurait pas d'autres possibilités intéressantes.

# 7 Bibliographie

[HLA08] HLADKY, Stephen & BULITKO, Vadim. *An evaluation of models for predicting opponent positions in First-Person shooter video games.* 2008, IEEE Symposium on computational intelligence and games

[SOU07] SOUTHEY, Finnegan, LOH, Wesley & WILKINSON, Dana. *Inferring complex agent motions from partial trajectory observations*. 2007, Hyderabad, India

[ROS10] ROSS, Sheldon M. *Introduction to probability models 10th edition*. 2010.

[MUR02] MURPHY, Kevin P. *Hidden semi-Markov models (HSMMs)*. 2002.