DEA Imagerie Vision Robotique 2002 / 2003

Apprentissage Bayésien de Comportements pour Personnages de Jeux Vidéos

Anthony Arrigoni

Rapport de DEA

Sous la direction de Pierre Bessière et Olivier Lebeltel.

Projet réalisé au sein de l'équipe SHARP,
au Laboratoire GRAphique, VIsion, Robotique
à INRIA Rhône-Alpes
ZIRST, 655 av. de l'Europe
38330 Montbonnot St. Martin
FRANCE

Table des matières

Introduction

L'intelligence artificielle dans les jeux videos a pendant longtemps été très décevante. En particulier, les jeux de tir à la première personne (First Person Shooter ou FPS), comme Wolfenstein, Doom ou Quake pour ne citer qu'eux, ne proposaient comme ennemis que des escouades de monstres suicidaires se jetant sur le joueur sans la moindre once d'intelligence. Mais cela a commencé à changer, les développeurs ont ressenti la nécessité d'offrir aux joueurs des adversaires plus intelligents. Plus de moyens ont été consacrés à la création de personnages intelligents tant au niveau du développement, que du calcul processeur. Et ainsi des jeux comme Half-Life ou UnrealTournament sont apparus, et ont surpris bon nombre de joueurs en proposant des IAs convaincantes. Mais beaucoup de chemin reste à parcourir dans ce domaine, et les acteurs du jeu vidéo lorgnent déjà du coté de la recherche et s'intéressent beaucoup à de nouvelles avancées qui pourraient les démarquer de la concurrence en leur fournissant des personnages beaucoup plus intelligents, qui pourraient s'adapter, et si possible apprendre au contact du joueur. Même si de nombreuses techniques d'intelligence artificielle et d'apprentissage ont été dévelloppées dans le milieux académiques, celles-ci ne s'implémentent pas ou peu dans le domaine des jeux vidéos dont les contraintes diffèrent de la recherche.

Les développeurs n'étaient pas très optimistes quant-aux capacités d'apprentissage des IA dans les jeux vidéos mais d'astucieux succès comme Black&White ont apporté un regain d'intérêt de la part de la profession. Le développement des jeux massivement multijoueur y est aussi pour quelque chose. Une IA ayant appris à jouer comme un être humain pourrait alors astucieusement le remplacer lorsque celui-ci doit s'absenter lors d'une partie éffrénée. Apprendre la stratégie de l'adversaire pourrait aussi permettre de le contrer car aujourd'hui, le jeu statique des personnages controlés par l'ordinateur est rapidement défait par le joueur.

Le but de ce DEA est d'effectuer un apprentissage du modèle comportemental du joueur et de programmer un personnage synthétique (manipulé par la machine) qui pourra jouer suivant ce modèle. Pour ce faire nous explorons l'utilisation de la programmation Bayésienne dans le domaine des jeux vidéos et tout particulièrement les capacités d'apprentissage que nous pouvons en tirer. Nous espérons prouver qu'il est possible de faire apprendre à un personnage synthétique à jouer comme un joueur en observant celui-ci, et ainsi obtenir un personnage performant et fidèle à l'original.

Dans un premier temps, nous exposerons la problématique de l'intelligence artificielle dans les jeux vidéos, son intérêt pour la recherche, et une courte revue des méthodes employées dans l'industrie. Nous examinerons ensuite la méthode de programmation bayésienne de personnage utilisée dans l'apprentissage, ainsi que ses avan-

tages et inconvénients principaux. Enfin nous détaillerons la méthode utilisée pour l'apprentissage ainsi que le protocole expérimental utilisé pour valider notre méthode.

Chapitre 1

Problématique de l'IA dans les jeux vidéos

1.1 Les jeux vidéos comme plateforme de recherche

1.1.1 Quelques mots sur l'IA dans la recherche et l'industrie

Les algorithmes appartenant au champ de l'intelligence artificielle sont largement répandus dans la recherche et dans de nombreuses disciplines (Robotique, Vision, etc). En particulier l'apprentissage y est régulièrement mis en oeuvre avec de nombreuses techniques différentes comme par exemple les réseaux de neurones, les machines à support de vecteur, les arbres de décision ou l'apprentissage bayésien. Mais ces méthodes ne sont pas ou peu utilisées par les développeurs de jeux. Les reproches qui leur sont souvent faits par les industriels sont la gourmandise en temps de calcul et la difficulté de mise en oeuvre. Leur flexibilité, leur expressivité et leur capacité d'apprentissage en font pourtant des approches efficaces et intéressantes.

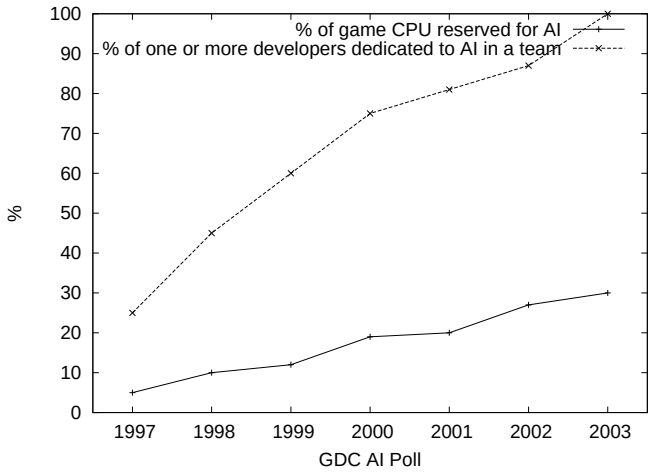
1.1.2 Quelle place pour les jeux vidéos dans la recherche ?

Les jeux vidéos sont aujourd'hui une plateforme utilisable et utilisée par la recherche. En effet, ils fournissent des problèmes mettant en jeu des interactions complexes et variées avec le joueur humain. Les personnages synthétiques peuvent jouer de multiples rôles ([LVL00]) comme adversaire ou partenaire du joueur humain, personnage secondaire, conseiller ou simple agent du jeu. Le but est bien évidemment d'avoir un personnage qui agit de la façon la plus humaine possible. L'idée d'utiliser les jeux vidéos pour créer des personnages synthétiques passant pour des êtres humains a été développée dans [Lai02], et elle n'est pas sans rappeler l'idée du test de Turing [Tur50], à savoir dans ce cas : peut-on créer un joueur synthétique qui fasse illusion aux yeux d'un humain ? Une telle approche est suggérée dans [LD00], où le critère d'humanité est mis en avant pour l'évaluation de personnages synthétiques.

1.2 L'IA dans les jeux vidéos

Depuis environ cinq ans, l'intelligence artificielle est en pleine expansion dans les jeux vidéos. L'arrivée des cartes 3D a permis d'alléger la charge du processeur et de libérer celui-ci des calculs de rendu graphique. De plus en plus de programmeurs sont affectés au développement de la partie intelligence artificielle (cf 1.1 : évolution de la répartition des ressources). Tout cela se conjugue à la pression des joueurs qui commencent à être satisfaits des performances graphiques mais largement mécontents des performances des adversaires synthétiques, et conduit à une forte demande et à un besoin de développement important dans ce domaine.

FIG. 1.1 – Évolution du temps et de la puissance dévolus à l'IA dans les jeux vidéos relevé à la GDC (Game Developers Conference)



1.2.1 Techniques employées

Les techniques suivantes sont les techniques principalement employées dans les jeux vidéos d'aujourd'hui. On pourra remarquer l'absence de certaines méthodes bien implantées dans la recherche comme les réseaux de neurones ou les algorithmes génétiques. Ces principes sont peu utilisé dans les pratiques industrielles [Woo01], car ils sont jugés peu prédictibles et difficiles à maîtriser.

Les systèmes de scripts

C'est une méthode simple, très répandue et qui s'adapte bien aux séquences d'actions linéaires. Généralement, un script ressemble à un programme classique avec des cascades de branchements conditionnels. Leur simplicité, les rend relativement faciles à mettre en oeuvre et faciles à maîtriser mais cela est fait au détriment de la richesse de programmation du joueur artificiel.

Les scripts sont très utiles pour programmer des enchaînements d'action comme par exemple pour suivre une ligne scénaristique mais permettent mal la programmation de comportements haut niveau.

Les automates

Les développeurs de jeux lorsqu'ils doivent modéliser des comportements plus complexes, difficiles à gérer avec des scripts utilisent généralement des machines à états finis (Finite State Machine ou FSM). Un automate à état finis est un ensemble d'états reliés par des transitions. Chaque état correspond à un mode de fonctionnement ou comportement, et les transitions sont franchies suivant certaines conditions logiques sur les capteurs et les actions.

Les automates fournissent un schéma simple qui permet de combiner différents comportements et les transitions entre eux, ce qui les rend très appréciables auprès des développeurs ; malheureusement, cette structure statique et les transitions déterministes se prêtent mal à l'apprentissage et peuvent devenir prévisibles. Cette technique est par exemple utilisée dans Unreal Tournament et Half-Life pour modéliser des adversaires humains convaincants.

Les automates sont aussi utilisés dans leur version floue (FuSM) ou non déterministe, ils gagnent alors en expressivité, au prix d'une complication du formalisme. Ils sont cependant sujets à l'explosion combinatoire du nombre de cas à traiter au niveau des transitions quand le nombre de variables augmente.

Autres méthodes

D'autres méthodes ont été utilisées dans le domaine des jeux vidéos mais sont minoritaires.

On peut par exemple citer la méthode dite de "flocking" [Rey87] qui est utilisée pour modéliser les comportements de groupe. Elle a l'avantage d'être relativement simple à programmer et de donner des résultats efficaces. Elle est par exemple utilisée dans les jeux Unreal et Half-Life et plus généralement dans les jeux de stratégie temps réel (RTS).

Une autre technique dite des objets intelligents (smart objects) mérite de retenir notre intérêt. Elle a été décrite dans [KT99] a été développée dans le jeux Les Sims. Elle est appelée le Terrain intelligent (smart terrain) par le créateur du jeu, Will Wright. Les objets du monde où évolue l'IA diffusent dans l'univers du jeu des messages à l'intention des personnages synthétiques expliquant leur fonction et comment les utiliser. Cela permet une grande souplesse et une grande extensibilité.

L'apprentissage dans les jeux vidéos

Finalement dans le domaine plus particulier de l'apprentissage, il convient de s'attarder sur le jeu Black&White. Le joueur y joue le rôle d'un dieu et influence le développement de l'univers, tout particulièrement le comportement de créatures qui apprennent grâce à une récompense ou une punition qui indique si les actions de la

créature sont bonnes ou mauvaises. Grâce à l'utilisation d'arbres de décision, la créature peut apprendre ce qu'elle doit faire et ne pas faire. Cette démonstration de l'utilité et de l'efficacité de l'apprentissage dans les jeux vidéos a déclenché le vif intérêt que portent aujourd'hui les développeurs à l'apprentissage.

D'autres jeux qui sont plutôt des démonstrations technologiques que des succès commerciaux utilisent l'apprentissage comme la série des Créatures (Creatures/Creatures 2/Creatures 3/Creatures Adventures). Ces jeux utilisent des réseaux de neurones pour simuler une créature et la faire apprendre. Ils utilisent également les algorithmes génétiques pour simuler les changements entre les générations. Il faut toutes fois noter l'importante puissance de calcul requise par chaque agent. Cela empêche ces méthodes d'être utilisées dans d'autres jeux commerciaux.

1.3 Intérêt de l'apprentissage supervisé par un joueur

1.3.1 Apprentissage par récompense

Certains jeux utilisent déjà le joueur pour faire de l'apprentissage. Ces jeux se servent généralement du joueur comme d'un juge qui donne une note, une récompense ou une punition à un agent qui apprend alors quelles sont les actions positives et négatives. C'est ce qui est par exemple utilisé dans Black&White ou Creature. Cet apprentissage rencontre un vif intérêt auprès des joueurs qui ont l'impression de modeler ou d'élever eux même l'agent apprenant.

1.3.2 Apprentissage par observation

Un deuxième type d'apprentissage supervisé par le joueur soulève un intérêt croissant auprès des développeurs de jeux, c'est l'apprentissage par observation du joueur. Ce type d'apprentissage est peu, voire pas du tout utilisé dans le domaine mais les développeurs en voient parfaitement l'intérêt. En effet, un adversaire synthétique qui apprendrait en observant son adversaire pourrait s'adapter aux stratégies du joueur et renouveler l'intérêt du jeu. Pour le moment, on observe plutôt le phénomène inverse : les adversaires synthétiques jouant toujours de la même façon, le joueur s'adapte au jeu de l'ordinateur qui est statique et apprend des stratégies qui permettent de le battre. Ces tactiques sont souvent inutiles contre un joueur humain car celui-ci change de stratégie et s'adapte rapidement.

Ce type d'apprentissage intéresse également beaucoup les créateurs de jeux massivement multijoueurs. En effet si lors d'une partie, un des participants doit s'absenter, quelques minutes, le jeu continue et souvent ces quelques minutes d'inactivité peuvent être fatales au joueur. Il suffirait alors de faire jouer pendant ce temps une IA qui aurait appris à jouer comme le joueur en l'observant. On pourrait également utiliser cette capacité pour s'entrainer contre soi-même et ainsi apprendre ses faiblesses, et comment y pallier.

1.4 Méthode d'apprentissage choisie

Nous mettons en oeuvre une méthode d'apprentissage par observation hors ligne dont le but est de laisser jouer l'être humain de façon naturelle, c'est à dire en utilisant uniquement les contrôles du jeu, et de faire ensuite évoluer un agent artificiel qui aura appris en observant la partie précédente et qui pourra restituer un comportement semblable à celui du joueur. Nous allons utiliser la programmation bayésienne pour modéliser les comportements. Après avoir observé la partie d'un être humain nous calculerons des distributions de probabilités qui nous permettrons de programmer un agent artificiel bayésien.

Chapitre 2

La programmation bayésienne de personnage

Pour l'apprentissage et la restitution du comportement appris, nous avons utilisé le jeu de combat à la première personne Unreal Tournament, augmenté de la modification Gamebots. Un jeu de combat à la première personne ou FPS (First Person Shooter) est une simulation de combat plus ou moins réaliste dans un environnement virtuel 3D où la caméra est placée en vue subjective, c'est à dire où l'on voit ce que verrait le personnage. Les personnages synthétiques (les adversaires humanoïdes contrôlés par la machine aussi appelés bots) que nous faisons évoluer utilisent un modèle de programmation bayésienne qui a été développé par Ronan Le Hy lors de son DEA en 2002 [LeHy02].

Nous allons brièvement présenter les spécifications et le mode de fonctionnement du bot bayésien ; pour plus de détail, j'invite le lecteur à se reporter au DEA de Ronan Le Hy.

2.1 La plateforme d'expérimentation

2.1.1 Le jeu

Nos personnages évoluent dans un univers tridimensionnel, et le joueur contrôle l'un d'eux en vue subjective. Le but du jeu est de tuer un maximum de fois les autres personnages (joueurs ou bots). Il dispose pour cela d'armes et de munitions pour se battre, d'armures et de vie pour se protéger, ainsi que des bonus divers (invisibilité, amplificateur de dommages). Chaque personnage lorsqu'il meurt ou commence la partie apparaît à un des points de départ de la carte avec deux armes, 100 points de vie et 0 point d'armure. La figure 2.1 montre un exemple de scène du jeu.

On peut donc distinguer deux composantes qui caractérisent le jeu d'un joueur : l'une de dextérité, à savoir viser juste, sauter, courir pour éviter les tirs, etc et l' autre plus stratégique de gestion des ressources, à savoir aller chercher des armes fortes, des armures, savoir quand fuir, etc.

FIG. 2.1 – Un bot en mauvaise posture lors d'une partie

2.1.2 Contrôle logiciel des personnages

Unreal Tournament supporte un langage orienté objet : les scripts Unreal. Ils permettent d'apporter de nombreuses modifications au jeu comme de nouvelles armes, de nouveaux modes ... L'interface entre le bot et le jeu, appelée Gamebot ([AMS+01], [KVS+02]) est basée sur ce système de script. Elle nous fournit une façon simple de nous connecter au moteur du jeu, en nous transmettant les perceptions du bot et en recevant les ordres et les commandes faites au bot. Nous avons mis en place une modification similaire pour recevoir les informations sensorielles que perçoit le joueur.

2.2 Programmation Bayésienne de Robots

Nous faisons ici un très bref rappel du principe de programmation bayésienne des robots. Pour une approche complète voir [Leb99], [BDL+98a], [BDL+98b] dont est tiré ce résumé.

2.2.1 Programme bayésien

Un programme bayésien est composé d'une question et d'une description. La description est constituée des données (résultat d'expériences) et des connaissances préalables. Les connaissances préalables sont spécifiées par trois éléments :

- les variables pertinentes ;
- la décomposition de la distribution conjointe qui spécifie les relations entre variables et les indépendances conditionnelles ;
- les formes paramétriques apparaissant dans la décomposition.

La question consiste à interroger la distribution en faisant l'inférence d'une distribution particulière, à partir de laquelle une décision peut être prise (par tirage par exemple)

$$\text{Programme} \begin{cases} \text{Description} \begin{cases} \text{Connaissances Préalables} \begin{cases} \text{Variables pertinentes} \\ \text{Décomposition} \\ \text{Formes paramétriques} \end{cases} \\ \text{Données} \end{cases} \\ \text{Question} \end{cases} \tag{2.1}$$

Nous allons maintenant détailler les variables, la décompositions et les formes paramétriques choisies pour le bot bayésien.

2.2.2 Description pour le bot bayésien

Variables pertinentes

Les variables pertinentes sont divisées en deux parties, les variables sensorielles qui résument la représentation du monde du bot et les variables d'état qui décrivent l'état dans lequel se trouve le bot. Pratiquemment ce qu'on cherche à faire c'est : connaissant la valeur des variables sensorielles et de l'etat à t, calculer la probabilité d'être dans un certain état à t+1. Connaissant cette probabilité on choisit dans quel état on va se trouver à t+1.

Les variables d'états sont au nombre de trois
- E_t = A, RA, RV, Ex, Fu, DD : État du bot à l'instant t
- E_{t+1} = A, RA, RV, Ex, Fu, DD : État du bot à l'instant t+1
- Tir = Vrai, Faux : indique si le bot doit tirer ou non à t+1.

Les valeurs que peuvent prendre les variables E_t et E_{t+1} sont
- A : Attaque, le bot recherche une cible et l'attaque
- RA : RechercheArme, le bot part à la recherche d'une arme
- RV : RechercheVie, le bot part à la recherche d'un bonus de santé
- Ex : Exploration, le bot explore le niveau
- Fu : Fuite, le bot cherche à s'éloigner des ennemis en présence
- DD : Détection Danger, le bot tourne sur lui même pour détecter la présence d'un éventuel danger

Les variables sensorielles sont plus nombreuses,
- Vie = Bas, Moyen, Haut : Le niveau de vie du bot
- Arme = Bas, Moyen, Haut : La puissance de l'arme du bot
- ArmeAdversaire = Bas, Moyen, Haut : La puissance de l'arme d'un éventuel adversaire
- Bruit = Vrai, Faux : Indique si le bot a entendu du bruit
- NombreEnnemis = Aucun, Un, DeuxOuPlus : Indique le nombre d'ennemis en présence
- ProxArme = Vrai, Faux : indique la proximité d'un arme
- ProxSante =Vrai, Faux : indique la proximité d'un bonus de santé

Décomposition

On a la distribution conjointe suivante :

$$P(E_t E_{t+1} Vie Arme Arme Adversaire Bruit NombreEnnemis ProxArme ProxSante Tir) \tag{2.2}$$

Pour la décomposer, nous allons utiliser les hypothèses d'indépendance conditionnelle suivantes :

- Les variables perceptives sont indépendantes deux à deux sachant E_{t+1}.
  La connaissance l'état de l'état du joueur résume complètement les informations contenues dans les autres variables sensorielles. Cette hypothèse n'est pas vraie pour toutes les variables ; en effet si on prend le niveau de vie et le niveau d'armure, ces deux variables ne sont clairement pas indépendantes, le niveau d'armure étant même parfois équivalent au niveau de vie. Cela a conduit à faire des choix comme regrouper ces deux valeurs au sein d'une même, la santé. Ainsi les variables choisies sont indépendantes conditionnellement à l'état du joueur.
- Les variables NombreEnnemis et Tir ne dépendent pas des autres.
  Le nombre d'ennemis ne dépend pas de des autres variables sensorielles propres au bot comme son niveau de vie ou son arme, mais plutôt des aléas du jeu. De même le fait de tirer ou pas dépend uniquement de la présence ou non d'ennemis.

Nous obtenons alors la décomposition suivante :

$$\begin{aligned} & P(E_t E_{t+1} Vie Arme Arme Adversaire Bruit NombreEnnemis ProxArme ProxSante Tir) \\ = \; & P(E_t) P(E_{t+1}|E_t) P(Vie|E_{t+1}) P(Arme|E_{t+1}) P(ArmeAdversaire|E_{t+1}) \\ & P(Bruit|E_{t+1}) P(NombreEnnemis|E_{t+1}) P(ProxArme|E_{t+1}) \\ & P(ProxSante|E_{t+1}) P(Tir|NombreEnnemis) \end{aligned} \tag{2.3}$$

Formes paramétriques

- P(Et) est choisie uniforme, on ne se donne en effet pas d'apriori sur l'état dans lequel se trouve le bot. En pratique cette variable se trouve toujours en partie droite, le choix n'est donc pas nécessaire pour faire l'inférence de notre question.
- Les autres probabilités sont spécifiées dans des tables

Identification

L'identification consiste à remplir les tables de probabilités. Elle est faite lors de l'apprentissage. La méthode utilisée pour remplir les tables sera expliquée au chapitre suivant.

2.2.3 Question

Pratiquement on cherche à déterminer l'état dans lequel va se trouver notre bot (E_{t+1}) connaissant tout le reste (état actuel et variables sensorielles). Ce qui nous

donne la question probabiliste suivante :

$$P(E_{t+1}|E_t VieArmeArmeAdversaireBruitNombreEnnemisProxArmeProxSante) \tag{2.4}$$

Elle est résolue de la façon suivante

$$\begin{aligned}
&P(E_{t+1}|E_t VieArmeArmeAdversaireBruitNombreEnnemisProxArmeProxSante) \\
&= \frac{P(E_{t+1}E_t VieArmeArmeAdversaireBruitNombreEnnemisProxArmeProxSante)}{P(E_t VieArmeArmeAdversaireBruitNombreEnnemisProxArmeProxSante)} \\
&= \frac{1}{Z}(P(E_t)P(E_{t+1}|E_t)P(Vie|E_{t+1})P(Arme|E_{t+1})P(ArmeAdversaire|E_{t+1}) \\
&\quad P(Bruit|E_{t+1})P(NombreEnnemis|E_{t+1})P(ProxArme|E_{t+1})P(ProxSante|E_{t+1})) \\
&= \frac{1}{Z'}\Big(P(E_{t+1}|E_t)P(Vie|E_{t+1})P(Arme|E_{t+1})P(ArmeAdversaire|E_{t+1}) \\
&\quad P(Bruit|E_{t+1})P(NombreEnnemis|E_{t+1})P(ProxArme|E_{t+1})P(ProxSante|E_{t+1})\Big)
\end{aligned}$$

$$\begin{aligned}
\text{avec } Z &= \sum_{E_{t+1}} P(E_t)P(E_{t+1}|E_t)P(Vie|E_{t+1})P(Arme|E_{t+1}) \\
&\quad P(ArmeAdversaire|E_{t+1})P(Bruit|E_{t+1}) \\
&\quad P(NombreEnnemis|E_{t+1})P(ProxArme|E_{t+1}) \\
&\quad P(ProxSante|E_{t+1}) \\
&= P(E_t)Z'
\end{aligned}$$

2.3 Discussion sur le bot bayésien

Le bot bayésien peut être comparé à un automate d'états finis dont les états seraient A, RV, RA, Fu, Ex, DD mais dont les transitions seraient complexes car calculées par des distributions de probabilité. Nous allons discuter les avantages et les inconvénients de ce modèle et le comparer à celui des bots d'Unreal Tournament qui sont de simples machines à états finis.

2.3.1 Avantages du bot bayésien

Les avantages du bot bayésien proviennent des transitions probabilistes. Étant non déterministes, elles ajoutent un caractère imprévisible au bot, le joueur ne peut donc pas apprendre la stratégie du bot et prévoir ses actions, mais le bot agit tout de même de façon plausible.

Le bot calcule ses probabilités de transition en fonction de tables spécifiant les probabilités conditionnelles de la décomposition présentée en 3.2. On peut programmer différents comportements en jouant sur ces tables. Le bot est donc ajustable, et ce en temps réel puisque les tables peuvent être changées en cour de partie.

C'est également grâce à ces tables que nous allons pouvoir faire de l'apprentissage. On cherche alors à déterminer les tables en utilisant des valeurs échantillonnées sur un joueur humain.

Le bot bayésien est donc beaucoup plus flexible que les bots d'Unreal qui sont figés. Comme pour les bots Unreal on peut ajuster leur niveau de difficulté. Pour les bots natifs on joue sur les dégâts, la rapidité du bot, son temps de réaction, sa précision sa tendance à bouger, sauter, éviter les tirs lorsqu'il est en mode combat. Si l'on désire modifier la difficulté du bot bayésien, on a deux possibilités :

- joueur sur le niveau de difficulté dans le jeu. Le bot bayésien est aussi soumis aux modifications que subissent les bots natifs. Cela ne jouera que sur la vitesse et la quantité de dégâts infligé par le bot bayésien et il n'y aura que deux niveaux différents ;
- on peut également modifier les modèles de comportement (on verra lors de test ultérieur que les performances peuvent varier suivant les tables du bot).

2.3.2 Inconvénients du bot bayésien

Le nombre d'état limité, le faible nombre de variables sensorielles, ainsi que leur discrétisation grossière influent lourdement sur la finesse de programmation du bot. Si on compare le modèle bayésien au bot d'Unreal, celui ci possède une vingtaine d'état, en comparaison les 6 états du bot bayésien font peu. En particulier l'état attaque est décomposé en 5 états dans le bot Unreal (Attacking, Charging, Hunting, TacticalMove, RangeAttack). La même limitation peut être constatée pour les variables sensorielles qui sont plus nombreuses, plus fines et mieux utilisées dans Unreal (cela provient de l'intégration du bot). Mais on pourra tout même noter qu'aucun des deux bot ne triche. Un dernier point qu'il convient de remarquer est que l'efficacité du bot bayésien est directement dépendante du codage des états. Mais tous ces points ne sont que des désavantages liés à l'implémentation et ne sont pas intrinsèques à notre forme de modèle bayésien.

Chapitre 3

Apprentissage d'un comportement

Notre plateforme de test permet le développement de plusieurs stratégies de jeu. Un joueur peut être très agressif, attaquer sans relâche, sans faire attention ni à sa vie, ni à son arme, ni à l'arme que possède son adversaire. Un autre joueur peut au contraire n'attaquer que quand il s'estime suffisamment équippé et se retirer du combat quand il est affaibli. C'est cela que nous désirons apprendre tout en laissant l'apprentissage transparent du point de vue du joueur, c'est à dire en le laissant utiliser les contrôles naturels du jeu. Comme nous l'avons vu au 3.2.2, l'apprentissage se fait en remplissant les tables spécifiant les probabilités conditionnelles qui apparaissent dans la décomposition (ex : $P(Vie|E_{t+1})$). Pour calculer les probabilités dans les tables, on a besoin de connaître la valeur de chacune des variables à chaque échantillon de temps (10 par secondes). Alors que les variables sensorielles peuvent être mesurées directement, il se pose le problème de déterminer l'état du joueur ($E_t et E_{t+1}$).

3.1 Apprentissage par sélection de comportements

Cet apprentissage évite la difficulté de reconnaître le comportement du joueur. En effet, dans cette méthode l'humain n'est plus joueur mais plutôt expert. Il doit observer la situation d'un bot lors d'une partie et lui fournir le comportement adéquat. On pilote le bot grâce à une interface (voir figure 3.1) qui permet d'observer en temps réel la situation du personnage et de lui imposer un comportement à effectuer.

On dispose facilement de la valeur des variables sensorielles (niveau de vie, arme, etc) et on connaît l'état du bot puisque c'est l'expert qui le spécifie. Ensuite il ne reste plus qu'à estimer les probabilités ; on réalise cette opération avec une loi de succession de Laplace (voir 3.3).

Nous allons utiliser cet apprentissage pour le comparer au nôtre et valider notre méthode de reconnaissance de comportement du joueur. L'avantage de cette méthode est qu'on connaît le comportement avec exactitude car celui-ci est imposé par l'expert, mais l'être humain perçoit avec difficulté le monde virtuel car son immersion est limitée par l'interface.

FIG. 3.1 – interface de pilote du bot

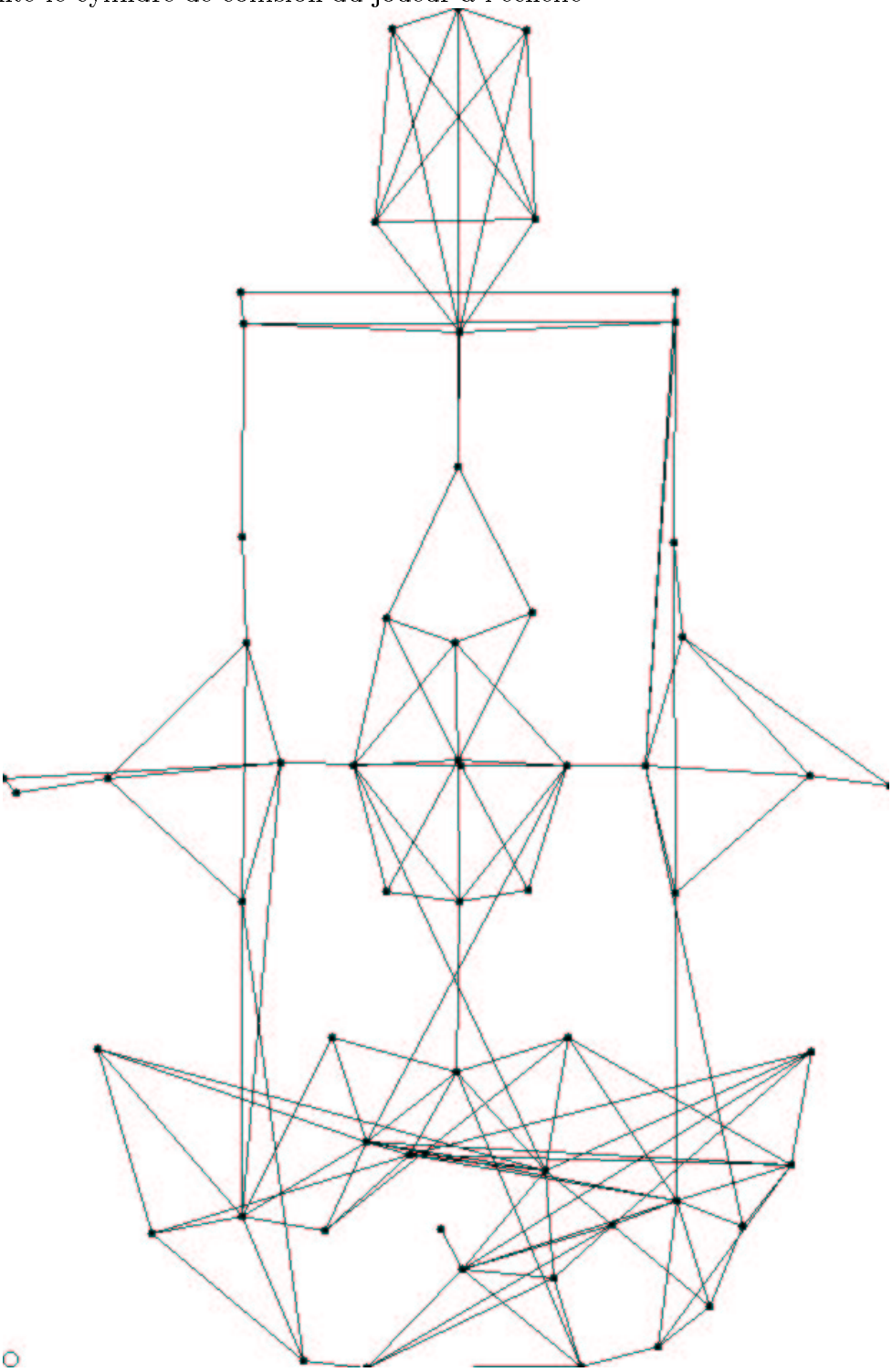
3.1.1 Le graphe de navigation

Le graphe de navigation est un graphe constitué de noeuds et de chemins reliant ces noeuds. Il est utilisé par les bots pour se déplacer dans le niveau et pour obtenir des informations intéressantes sur ce niveau. Les principaux noeuds sont les suivants :

- PathNode : Le point de navigation le plus simple.
- PlayerStart : Indique où les joueurs et les bots peuvent apparaître.
- InventorySpot : Point où peut se trouver un objet d'inventaire (Vie, armes, munitions, ...).
- AmbushPoint : Point où on peut tendre une embuscade.
- LiftCenter, LiftExit : Utilisé pour permettre aux bots d'utiliser les ascenseurs.
- JumpSpot : Indique aux bots où utiliser les bottes de saut. Les bottes de saut sont un objet de l'inventaire que l'on peut utiliser 3 fois et qui permettent de sauter beaucoup plus haut et donc d'atteindre des points inaccessibles autrement.
- TranslocStart, TranslocDest : Indiquer aux bots où utiliser le translocateur. Le translocateur est un objet qui s'utilise comme une arme, on lance une balise et on peut ensuite se téléporter à l'endroit de cette balise.
- VisibleTeleporter, Teleporter : Indiquer un téléporteur et sa destination.
- DefensePoint, FlagBase, AlternatePath, ... : Différents points utilisés dans les modes par équipe (capturer le drapeau, domination).

Le graphe est totalement invisible au joueur mais certains noeuds sont connus du joueur comme les noeuds d'inventaire, et d'apparition dans le niveau. On peut donc en déterminant la position du joueur sur ce graphe connaître son déplacement par rapport aux différents bonus et armement et donc savoir si le joueur se dirige vers de la vie ou une arme. Mais le joueur peut effectuer des mouvements illégaux et ainsi passer d'un noeud à un autre alors qu'aucun lien ne relie ces deux noeuds. Un exemple de graphe de navigation est donné par la figure 3.2 qui est tirée d'un niveau très simple (2-3 joueurs).

FIG. 3.2 – Le graphe de navigation du niveau DM-Oblivion. Le cercle en bas à gauche représente le cylindre de collision du joueur à l'échelle



3.2 Déterminer l'état du joueur

Le problème de reconnaissance de l'état du joueur peut être énoncé ainsi, connaissant les informations reçues par le joueur (présence d'ennemi, bruits entendus, etc) et les commandes bas niveau envoyées par le joueur à son avatar virtuel (Tirer, reculer, tourner, etc) on veut retrouver la logique qui a dicté ces actions et donc quel était le comportement du joueur parmis ceux connus du bot (Attaque, Fuite, RechercheVie, etc). Pour déterminer le comportement du joueur et faire jouer ce comportement de façon crédible, nous avons besoin de plaquer les actions du joueurs sur les actions du bot bayésien dans un état déterminé.

3.2.1 Comparaison entre le bot bayésien et le joueur

Pour chaque état du bot, nous avons effectué une analyse du comportement du bot et nous avons estimé quel est le comportement du joueur dans un tel état. Cela nous a permis de déterminer les variables intermédiaires intéressantes pour choisir l'état du joueur. Cela a aussi mis en avant les problèmes qui se posent pour déterminer les différents états, et les différences entre les comportements de joueur humain et du bot.

L'attaque

Lorsque le bot est en attaque, il choisit une cible aléatoire parmi ses adversaires et tente d'accorder sa distance à la cible. La distance qu'il va tenter de maintenir va dépendre de l'arme : il peut tenter d'être plus loin qu'une certaine distance pour les armes de très longue portée ; il peut tenter de se maintenir autour d'une distance pour les armes de moyenne portée ; il peut vouloir être au dessous d'une certaine distance pour les armes de corps à corps.

Le joueur, quant à lui choisit une cible non aléatoire, celle qui parait le plus forte ou la plus près par exemple, et essaye d'accorder sa distance avec la cible. La distance dépend bien entendu de l'arme mais aussi du type de tir utilisé et du style de jeu du joueur.

De cette analyse on voit apparaître la nécessité d'utiliser la variable intermédiaire qui est la distance du joueur par rapport à sa cible ainsi que sa vitesse vers la cible.

La Détection du Danger

Lorsque le bot est en détection danger, il tourne sur lui même pour observer ce qui se passe autour de lui.

Aucun être humain ne se met à tourner sur lui même. Si un être humain cherche à observer ce qui se passe autour de lui, il va très rapidement regarder derrière lui (il effectue un demi-tour) puis il va revenir dans sa position initiale et regarder de l'autre coté. Balayer rapidement autour de soi suffit au joueur, de plus le contrôle à la souris l'empêche de faire un tour complet facilement et rapidement.

Nous avons donc besoin de connaître l'angle entre la position actuelle du joueur et celle ou il se trouvait il y a un court laps de temps. Nous avons choisi un intervale

de deux secondes (20 échantillons) car si on prend plus, on risque de détecter un état DetectionDanger lorsque le joueur tourne dans les couloirs et les intersections pour suivre la géométrie du niveau.

La fuite

Lorsque le bot est dans l'état fuite, il essaie de choisir un noeud du graphe de navigation qui va l'éloigner de ses ennemis.

Le joueur va courir dans une direction qui l'éloigne de ses plus proches ennemis mais va aussi utiliser le décor pour se cacher.

On a donc besoin de la variable intermédiaire qui est la vitesse par rapport à l'ennemi le plus proche.

La recherche de vie ou d'arme

Lorsque le bot est en recherche de vie ou d'arme, il utilise sa mémoire pour trouver un objet du type spécifié (arme ou vie), et se dirige vers le point où se situe l'objet.

Le joueur fait de même s'il connaît le niveau : il se dirige vers un point d'inventaire où un objet peut se trouver. Par contre s'il ne connaît pas la carte il va se déplacer au hasard jusqu'à ce qu'il trouve un objet.

Pour déterminer cet état nous avons besoin d'informations sur le niveau et sur les points intéressants. Nous allons utiliser le graphe de navigation qu'utilisent les bots (bayésien et d'Unreal) pour se déplacer.

Exploration

Lorsque le bot est en exploration, il choisit un des noeuds du graphe de navigation qu'il n'a pas visité et il s'y rend.

Si un joueur humain connaît le niveau, il ne fait pas d'exploration, il se déplace juste lorsqu'il recherche des bonus ou des ennemis. S'il ne connaît pas le niveau parfois, il explore le niveau, parfois il essaye de rester dans les endroits qu'il connaît. Mais le problème est qu'il peut oublier là où il est passé ou ne pas reconnaître l'endroit, ou être perdu. On ne peut donc savoir quand il explore et quand il n'explore pas.

On voit que l'état d'exploration est difficile à déterminer pour l'être humain. Dans la suite, cet état a été considéré comme étant l'état par défaut. C'est à dire que, si le joueur se déplace, tombe sur un ennemi et se met à attaquer, alors on supposera que les états précédents étaient de l'exploration

3.2.2 Génération de l'information de haut niveau

Nous avons vu dans la comparaison entre le bot bayésien et le joueur qu'on peut déterminer l'état équivalent d'un bot bayésien en observant certaines variables intermédiaires. On calcule donc avec les informations bas niveau les variables suivantes :

- la position du joueur sur le graphe de navigation ;
- la distance entre le joueur et la cible ;

- la vitesse vers la cible ;
- la vitesse vers l'ennemi le plus proche ;
- l'amplitude de rotation du joueur au cours des 2 dernières secondes.

Ces variables intermédiaires vont nous permettre de retrouver l'état dans lequel se trouve le joueur. On va différencier plusieurs classes d'états :

- Etats à validation retardée (RechercheVie et RechercheArme).
  C'est seulement lorsque le joueur atteint le point de navigation correspondant (un InventorySpot avec un bonus de santé ou une arme) qu'on assigne à tous les états antérieurs inconnus la valeur de RechercheVie ou RechercheArme.
- Etat à validation fixe (DetectionDanger).
  Si dans les deux dernières secondes le joueur à fait un demi-tour on donne aux 20 enregistrements précédents l'état de DetectionDanger.
- Etats à validation instantanée (Fuite, Attaque).
  Si le joueur possède une cible (il vise un ennemi) et si sa vitesse par rapport à sa cible est en accord avec l'arme qu'il porte et sa distance, alors on détermine que cet état est Attaque.
  Si le joueur s'éloigne de l'ennemi le plus proche et qu'il n'attaque pas (pas de cible, ou vitesse incorrecte), on assigne à cet état la valeur de Fuite.
- Etat par défaut (Exploration).
  Tous les états non déterminés sont considérés comme de l'exploration. Cela arrive lorsqu'on détecte un nouvel état qui n'est pas à validation retardée ou lorsque le joueur meurt.

Nous avons ainsi défini la méthode de reconnaissance de comportement qui s'appuie sur les variables intermédiaires.

3.3 Calcul des tables de probabilité

Une fois que l'état du joueur a été déterminé, on connaît toutes les variables à chaque pas de temps, car les variables sensorielles sont directement évaluables à partir des données bas niveau. Il ne nous reste alors plus qu'à évaluer les probabilités conditionnelles qui nous intéressent, pour cela nous allons utiliser des lois de succession de Laplace. Ces probabilités sont ensuite écrites dans les tables. Si on cherche par exemple à remplir la table $P(V|W)$, à chaque pas de temps, on compte les occurrences de $(V = V_i, W = W_j)$ pour tous les V_i et les W_j possibles. Si n_{ij} est le nombre d'occurrences de l'événement $(V = V_i, W = W_j)$, la loi de succession de Laplace nous donne alors :

$$P(V = V_i|W = W_j = \frac{1 + n_{ij}}{\sum_i n_{ij} + |V|}) \tag{3.1}$$

Une des particularités des lois de succession de Laplace est que pour un nombre d'échantillons nuls, on obtient une distribution uniforme, et plus l'échantillon devient grand, plus la loi tend vers celle donnée par les fréquences :

$$\frac{1 + n_{ij}}{\sum_i n_{ij} + |V|} \sim_{n_{ij} \to +\infty} \frac{n_{ij}}{\sum_i n_{ij}}$$

Cette loi n'assigne jamais une probabilité nulle. Une discussion plus poussée de son intérêt peut être trouvé dans [Jay03].

On voit donc que, connaissant toutes les variables il suffit de compter les occurrences des événements qui nous intéressent (les $V = V_i, W = W_j$) pour évaluer les probabilités conditionnelles correspondantes et remplir les tables. Elles peuvent ensuite être chargées directement par le bot bayésien qui va alors rejouer le modèle comportemental du joueur.

Chapitre 4

Protocole expérimental

4.1 Capture des données

On fait jouer différents joueurs humains n'ayant aucune connaissance du modèle mais ayant simplement une expérience du jeu, pour que les contrôles lui soient familiers. De plus, le joueur doit connaître le niveau dans lequel on effectue les mesures car comme on l'a vu précédemment, si le joueur ne maitrise pas la carte jouée il faudra évaluer son apprentissage de la géométrie du niveau ce qui constitue une énorme difficulté. Par contre on peut utiliser n'importe quel niveau du jeu.

Pendant la capture des données, un expert observe le joueur et note les tendances générales du comportement du joueur. Cela va nous permettre de comparer les résultats de notre apprentissage avec l'expertise faite sur le joueur.

Finalement, on fait jouer le bot bayésien avec les tables apprises pour vérifier son efficacité. Pour évaluer ses performances, on le fait se battre contre différents types de bot, dont des bots natifs d'Unreal, un bot bayésien dont les tables sont uniformes, des bots dont les tables sont spécifiés à la main et un bot issu d'un apprentissage par sélection de comportement.

4.2 Détail des différents bots

Les bots qui ont été testés sont les suivants :

Le bot d'Unreal

Nous avons fait jouer un bot d'Unreal pour avoir un étalon du niveau des différents bots bayésiens. Les bots Unreal possèdent huit niveaux de difficulté qui sont dans l'ordre novice, moyen, expérimenté, habile, adepte, maître de guerre, inhumain, dieu. Le bot Unreal utilisé lors des tests est en niveau expérimenté.

Les bots appris par reconnaissance de comportements (notre méthode)

Nous avons fait jouer deux bots de ce type, l'un avec des tables apprises sur un joueur dont le comportement général est très agressif, et l'autre appris sur le même joueur, jouant cette fois-ci en utilisant un comportement prudent.

Un bot appris par sélection de comportements

On utilise les tables résultant d'un apprentissage par sélection de comportement qui a été décrit au 3.1 pour programmer ce bot bayésien. Elles ont été collectés pendant 10 à 15 min ce qui constitue un volume d'échantillons à peu près identique à celui recueilli pour les bots appris par reconnaissance de comportement.

Les bots avec des tables spécifiées à la main

On dispose également de deux comportements dont les tables ont été spécifiée à la main, dont un bot normal (appelé odge) et un bot très agressif (appelé brsk pour berserk). Ces tables ont été crées par Ronan Le Hy pour programmer deux comportements différents pour le bot bayésien.

Le bot uniforme

On utilise enfin un bot dont les tables sont remplies par des distributions uniformes sauf pour la table de transition entre les états dont on privilégie l'auto-maintien (tend à rester dans le même état). En effet si on met une table de changement d'état uniforme, le bot va sans cesse passer d'un état à un autre ce qui le conduit à faire n'importe quoi, il est alors complètement inefficace. Ce bot va nous servir à évaluer la validité de notre méthode.

4.3 Résultats au niveau des tables

Nous présentons ici les tables de probabilité apprises et, nous les comparons aux tables créées à la main. Ces tables correspondent à une partie simple en 50 points ce qui équivaut à peu près à 10 à 15 minutes de jeu. Les tables sont présentées ainsi : les probabilités sont normalisées en colonnes, et les x indiquent une même valeur dans la colonne. Une colonne de x indique donc une distribution uniforme.

4.3.1 $P(E_{t+1}|E_t)$

Sur la table de transition entre états ($P(E_{t+1}|E_t)$) des bots spécifés à la main (tableau 4.1), une probabilité élevée est spécifiée sur la diagonale pour privilégier la stabilité (auto-maintien). Le bot berserk a en plus une probabilité élevée de passer en attaque.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
A	x	x	x	x	x	x
RA	$1e-05$	x	$1e-05$	$1e-05$	$1e-05$	$1e-05$
RV	$1e-05$	$1e-05$	x	$1e-05$	$1e-05$	$1e-05$
Ex	$1e-05$	$1e-05$	$1e-05$	x	$1e-05$	$1e-05$
Fu	$1e-05$	$1e-05$	$1e-05$	$1e-05$	x	$1e-05$
DD	$1e-05$	$1e-05$	$1e-05$	$1e-05$	$1e-05$	x

odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
A	x	0.01	0.01	0.01	0.01	0.01
RA	0.01	x	0.01	0.01	0.01	0.01
RV	0.01	0.01	x	0.01	0.01	0.01
Ex	0.01	0.01	0.01	x	0.01	0.01
Fu	0.01	0.01	0.01	0.01	x	0.01
DD	0.01	0.01	0.01	0.01	0.01	x

TAB. 4.1 – $P(E_{t+1}|E_t)$

table apprise

La table de transition entre les états (tableau 4.2) ressemble fortement à la table entrée à la main de type odge. En effet on retrouve la forte tendance à l'auto-maintien grâce aux probabilités de transition fortes sur la diagonale (cellules gris foncé), mais on peut remarquer que en dehors de la diagonale, les probabilités ne sont pas uniformes. Cette table nous apporte donc une information supplémentaire sur les transitions entre les états. Par exemple, on voit qu'à partir de l'état Fuite, la probabilité de passer à l'état Exploration et plus élevée que de passer dans d'autres états (cellule gris claire). On a donc plus d'information que dans la table spécifiée à la main.

	A	RA	RV	Ex	Fu	DD
A	0.865	0.006	0.011	0.107	0.019	0.060
RA	0.004	0.968	0.004	0.003	0.007	0.030
RV	0.012	0.008	0.962	0.002	0.018	0.030
Ex	0.097	0.014	0.019	0.846	0.122	0.030
Fu	0.022	0.003	0.003	0.041	0.832	0.030
DD	0.001	0.002	0.002	0.001	0.001	0.818

TAB. 4.2 – $P(E_{t+1}|E_t)$

4.3.2 $P(Vie|E_{t+1})$

La table relative au niveau de vie $P(Vie|E_{t+1})$ (tableau 4.3) nous apprend que alors que le bot berserk est totalement indifférent à son niveau de vie, l'autre bot (odge) a des probabilités proches du sens commun : une probabilité élevée d'avoir un niveau de

vie bas lorsqu'il fuit ou lorsqu'il recherche de la vie, une probabilité élevée d'avoir un niveau de vie haut lorsqu'il attaque.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Bas	x	x	x	x	x	x
Moyen	x	x	x	x	x	x
Haut	x	x	x	x	x	x
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Bas	0.001	0.1	x	0.1	0.7	0.1
Moyen	0.1	x	0.01	x	0.2	x
Haut	x	x	0.001	x	0.1	x

TAB. 4.3 – $P(Vie|E_{t+1})$

table apprise

Dans la table apprise (tableau 4.4), on retrouve des caractéristiques identiques à la table spécifiée à la main version odge. Par exemple lorsqu'on est dans l'état fuite, la probabilité que la vie soit faible ou moyenne est beaucoup plus élevée que celle d'être forte. Mais on observe des divergences, deux phénomènes peuvent les expliquer :

- Un biais dû à un très grand nombre d'occurence d'une valeur de la variable est introduit, en effet la vie du joueur au cours de la partie est souvent moyenne (ligne gris clair) car le joueur apparaît avec une vie moyenne ;
- Certains comportements propres aux joueurs induisent ces différences. Par exemple la faible valeur de $P(Vie = Forte|E_{t+1} = RechercheArme)$ (cellule gris foncé) peut s'expliquer par le fait que lorsque le joueur à une vie forte, il possède déjà une bonne arme, la priorité de bon nombre de joueurs étant en effet de se procurer une arme puissante avant tout.

	A	RA	RV	Ex	Fu	DD
Faible	0.179	0.342	0.307	0.191	0.457	0.033
Moyen	0.478	0.647	0.508	0.486	0.395	0.933
Forte	0.343	0.011	0.185	0.323	0.148	0.033

TAB. 4.4 – $P(Vie|E_{t+1})$

4.3.3 $P(NombreEnnemis|E_{t+1})$

La table concernant le nombre d'adversaires $P(NombreEnnemis|E_{t+1})$ (tableau 4.5) spécifie que le bot odge a tendance à fuir lorsqu'il y a trop d'ennemis, alors que pour berserk plus il y a d'ennemis, plus il a de chance de passer en attaque. De plus les deux bots ont tendance à explorer lorsqu'ils n'ont pas d'ennemis en présence.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucun	0	x	x	x	x	x
Un	0.2	x	x	$1e-09$	x	$1e-09$
DeuxOuPlus	0.8	x	x	$1e-10$	x	$1e-10$
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucun	0	x	x	x	0	x
Un	x	0.1	x	0.001	0.1	0.001
DeuxOuPlus	0.1	0.01	x	0.0001	x	0.0001

TAB. 4.5 – $P(NombreEnnemis|E_{t+1})$

table apprise

Dans cette table (tableau 4.6), on retrouve quelques similitudes avec les tables spécifiées à la main. Par exemple la probabilité d'être en DetectionDanger ou RechercheArme et d'avoir des ennemis est très faible (colonne gris clair), mais aussi celle d'être en RechercheVie (colonne gris foncé) contrairement au tables spécifiées à la main. La table apprise pour l'état Attaque fait beaucoup moins de différence entre la présence de un ou plusieurs joueur que celles spécifiées à la main.

	A	RA	RV	Ex	Fu	DD
0	0.001	0.919	0.829	0.367	0.001	0.900
1	0.696	0.071	0.121	0.426	0.781	0.067
2 ou +	0.304	0.009	0.050	0.206	0.217	0.033

TAB. 4.6 – $P(NombreEnnemis|E_{t+1})$

4.3.4 $P(ProxArme|E_{t+1})$ et $P(ProxSanté|E_{t+1})$

Ces deux tables (tableau 4.7 et 4.8) ont été construites pour rendre le bot opportuniste quand il passe à coté d'un bonus en excluant certain états.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	x	x	x	x	x
Vrai	x	x	$1e-05$	$1e-05$	x	$1e-05$
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	x	x	x	x	x
Vrai	x	x	$1e-05$	$1e-05$	x	$1e-05$

TAB. 4.7 – $P(ProxArme|E_{t+1})$

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	x	x	x	x	x
Vrai	x	$1e-05$	x	$1e-05$	x	$1e-05$
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	x	x	x	x	x
Vrai	x	$1e-05$	x	$1e-05$	x	$1e-05$

TAB. 4.8 – $P(ProxSante|E_{t+1})$

table apprise

Ces tables (tableau 4.9 et 4.10) se rapprochent de celles écrites à la main, tout en étant moins marquées. De plus la où les tables écrites à la main avaient une distribution uniforme, celles apprises font une différence.

	A	RA	RV	Ex	Fu	DD
Faux	0.839	0.863	0.508	0.826	0.881	0.103
Vrai	0.161	0.137	0.492	0.174	0.119	0.897

TAB. 4.9 – $P(ProxSante|E_{t+1})$

	A	RA	RV	Ex	Fu	DD
Faux	0.697	0.501	0.737	0.746	0.822	0.966
Vrai	0.303	0.499	0.263	0.254	0.178	0.034

TAB. 4.10 – $P(ProxArme|E_{t+1})$

4.3.5 $P(Arme|E_{t+1})$

La table $P(Arme|E_{t+1})$ (tableau 4.11) inhibe le comportement d'attaque et favorise le comportement de fuite et de recherche d'arme lorsque la puissance de l'arme est faible.

table apprise

On retrouve également dans cette table (tableau 4.12) de nombreuses caractéristiques de la table spécifiée à la main odge, par exemple la probabilité d'avoir une arme forte lorsqu'on attaque est plus forte que celle d'avoir une arme faible (colonne gris clair). Mais on observe également des différences qui on la même origine que celles évoquées pour la table $P(Vie|E_{t+1})$. C'est par exemple le cas pour la fuite : la probabilité est élevée lorsqu'on a une arme faible ce qui est logique mais aussi élevée lorsqu'on a une arme forte (cellule gris foncé), car les joueurs entrent très souvent en combat avec une arme forte.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucune	0.01	x	x	x	x	x
Moyenne	x	0.001	x	x	0.001	x
Forte	x	$1e-07$	x	x	$1e-05$	x
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucune	$1e-05$	x	x	x	x	x
Moyenne	x	0.1	x	x	0.1	x
Forte	x	$1e-05$	x	x	0.01	x

TAB. 4.11 – $P(Arme|E_{t+1})$

	A	RA	RV	Ex	Fu	DD
Faible	0.184	0.573	0.191	0.166	0.409	0.933
Moyen	0.291	0.254	0.382	0.279	0.127	0.033
Forte	0.525	0.173	0.427	0.555	0.463	0.033

TAB. 4.12 – $P(Arme|E_{t+1})$

4.3.6 $P(ArmeAdversaire|E_{t+1})$

Cette table (tableau 4.13) inhibe le comportement d'attaque si l'adversaire possède une arme puissante.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucune	x	x	x	x	x	x
Moyenne	x	x	x	x	x	x
Forte	0.1	x	x	x	x	x
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Aucune	x	x	x	x	x	x
Moyenne	x	x	x	x	x	x
Forte	0.01	x	x	x	x	x

TAB. 4.13 – $P(ArmeAdversaire|E_{t+1})$

table apprise

Les tables (tableau 4.14) sont cette fois-ci très différentes des tables spécifiées à la main. Plusieurs phénomènes entrent en jeu qui expliquent ces différences :

- Tout d'abord des résultats semblent intuitifs. Il semble en effet normal que dans l'état fuite la probabilité que l'adversaire possède une arme forte est élevée (cellule gris clair).
- Des informations supplémentaires apparaissent. Certains états sont souvent incompatibles avec la présence d'ennemis comme on peut le voir dans la table $P(NombreEnnemis|E_{t+1})$. Nous avons choisi de considérer que lorsqu'il n'y a

pas d'ennemis, la variable $ArmeAdversaire$ est Faible. Certaines probabilités $P(ArmeAdversaire = Faible|E_{t+1})$ élevées (cellules gris foncé) indiquent donc une absence d'ennemis.

- Le comportement des ennemis et du joueur influent sur les tables. Le fait que $P(ArmeAdversaire = forte|E_{t+1} = Attaque)$ soit élevée (cellule gris très foncé) est dû à la préférence d'armes fortes de la part des ennemis, mais aussi à la volonté du joueur d'en découdre malgré la puissance de son adversaire.

	A	RA	RV	Ex	Fu	DD
Faible	0.371	0.962	0.905	0.589	0.317	0.900
Moyen	0.133	0.009	0.004	0.109	0.146	0.033
Forte	0.496	0.029	0.090	0.302	0.537	0.067

TAB. 4.14 – $P(ArmeAdversaire|E_{t+1})$

4.3.7 $P(Bruit|E_{t+1})$

Cette table (tableau 4.15) décrit surtout l'exclusion entre certains états et la possibilité d'entendre du bruit.

brsk	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	0.1	0.1	$1e-05$	x	$1e-05$
Vrai	x	x	x	x	x	x
odge	*A*	*RA*	*RV*	*Ex*	*Fu*	*DD*
Faux	x	0.1	0.1	$1e-05$	x	$1e-05$
Vrai	x	x	x	x	x	x

TAB. 4.15 – $P(Bruit|E_{t+1})$

table apprise

Ces probabilités (tableau 4.16) sont très différentes de celles spécifiées à la main. Tout particulièrement lorsqu'on observe l'état DetectionDanger où elles sont très marquées mais complètement inversées (colonne gris clair). C'est en faisant une reconnaissance en ayant coupé le son que nous avons pu comprendre ce phénomène. En effet en coupant le son, les probailités (tableau 4.17) ressemblaient mieux à celles spécifiées à la main, particulièrement dans l'état DetectionDanger ou elles étaient quasiment identiques (colonne gris foncé). L'explication provient de la différence de perception entre le bot et le joueur. Lorsqu'un ennemi fait feu, le bot reçoit un message indiquant du bruit qui provient de l'arme mais aussi des impacts de balle sur les murs. Le bot ne sait donc pas où se trouve l'ennemi et doit passer en DetectionDanger. L'être humain peut différencier un son d'impact et celui d'une arme et grâce au son spatialisé il peut déterminer où est l'ennemi, il ne lui reste plus qu'à l'attaquer.

	A	RA	RV	Ex	Fu	DD
Faux	0.418	0.535	0.659	0.526	0.483	0.909
Vrai	0.582	0.465	0.340	0.474	0.517	0.091

TAB. 4.16 – $P(Bruit|E_{t+1})$ avec le son

	A	RA	RV	Ex	Fu	DD
Faux	0.227	0.374	0.259	0.274	0.228	0.138
Vrai	0.772	0.626	0.741	0.726	0.772	0.862

TAB. 4.17 – $P(Bruit|E_{t+1})$ sans le son

4.3.8 Conclusion sur les tables apprises

Ces tables ressemblent beaucoup à celles spécifiées à la main avec des probabilités souvent moins marquées. Cela paraît logique étant donné que les tables écrites à la main ont été fabriquées en utilisant le bon sens. Les différences observées peuvent avoir des origines multiples en dehors du bruit inévitable, à savoir : un comportement spécifique au joueur, des informations supplémentaires qui apparaissent, des choix de programmation, des différences perceptives entre le joueur et le bot ou des biais dans la quantité des données. L'idéal serait de pouvoir conserver les deux premières causes qui nous intéressent dans cette application et de supprimer les deux dernières qui dégradent nos performances.

4.4 Résultats au niveau des performances

Les bots appris sont efficaces, ils font jeu égal avec les bots spécifiés à la main et avec le bot d'Unreal de niveau expérimenté, sur les parties de test entre les différents bots, sur un score du vainqueur de 100, leurs scores varient en moyenne de +/- 10% par rapport aux autres. On peut noter que les versions aggressives (spécifiées à la main ou par apprentissage) sont en moyenne un peu plus performantes que les autres. Par contre les bots issus de l'apprentissage par sélection et les bots uniformes sont en moyenne environ 40% moins performants.

Quelques résultats de ces tests sont présentés au tableau 4.18, chaque ligne représente une partie en 100 points. Un point étant attribué à chaque fois qu'un personnage en tue un autre. Les X représentent des parties où les bots concernés n'ont pas jouées.

4.4.1 Analyse des résultats

Les résultats obtenus dans les tables correspondent globalement à nos attentes, elles reflètent assez fidèlement le comportement du joueur et produisent des bots efficaces

bot appris agressif	bot appris prudent	bot appris par sélection	bot spécifié normal	bot spécifié agressif	bot uniforme	bot Unreal
100	X	48	98	X	65	91
95	X	56	100	X	57	X
84	79	X	88	100	X	X
X	85	59	X	100	68	88
92	92	72	X	100	X	97
100	71	48	78	73	52	96
100	93	58	76	79	49	73
98	97	39	87	100	50	58

TAB. 4.18 – Test de performance des différents bots

aussi performant que les bots spécifiés à la main. On aurait pu penser que l'apprentissage par spécification de comportement serait plus performant que celui que nous avons effectué or c'est tout le contraire, l'apprentissage par spécification n'est pas plus performant qu'un bot uniforme, ce qui veut dire qu'il a été complètement inefficace. On peut expliquer ce résultat par une mauvaise appréciation par le joueur de l'environnement lors de l'apprentissage. Lorsque le joueur joue, l'immersion dans l'environnement virtuel lui procure une meilleurs vision du jeu, ce qui lui permet d'exploiter au mieux les informations venant du jeu.

Il apparaît des différentes tables apprises sur différents joueurs que ces tables se ressemblent beaucoup. Deux raisons peuvent expliquer ce phénomène.

Des prétraitements insuffisants

Les variables que nous avons choisies pour valider les états ne sont peut être pas adaptées et nous empêchent de déterminer avec justesse l'état du joueur, ce qui nous conduit indûment à des tables semblables. Il faudrait choisir des variables intermédiaires qui différencient plus les joueurs, ou des variables sensorielles et des variables d'état mieux adaptées.

Nécessité du jeu

Cette explication parait la plus plausible. L'observation des joueurs, nous fait remarquer qu'ils jouent tous de manière très agressive et souvent selon le même principe. Dès qu'ils apparaissent dans le niveau, ils vont chercher une arme forte le plus vite possible et ensuite ils cherchent des ennemis et se battent avec eux jusqu'à la mort. Les joueurs fuient rarement et ne vont chercher de la vie que s'ils n'ont rien d'autre à faire. Le jeu et son principe conduit à un comportement agressif, car c'est le plus efficace. Le plus souvent mourir n'est pas une pénalité car on réapparaît immédiatement avec sa vie de départ (100). Il est plus pénalisant de fuir car on perd beaucoup de temps à récupérer sa vie et on rate la possibilité de tuer des ennemis et donc de marquer un point.

C'est pour cela que nous avons réalisé un enregistrement où nous avons joué de façon prudente intentionnellement. On peut alors observer des différences notables entre les tables.

Conclusion

L'objectif de ce DEA était d'apprendre un comportement en observant un joueur. Différents problèmes techniques, comme en particulier la reconnaissance de comportement, se sont posés. La résolution de ces problèmes nous a permis de calculer les tables de probabilité associées au jeu d'un être humain et de programmer un bot bayésien avec les tables apprises. Le bot rejouant le comportement s'est avéré aussi performant que des bots dont les tables sont spécifiées à la main par le programmeur, et que des bots Unreal en difficulté expérimenté (difficulté 3 sur une échelle de 8). On peut donc dire que cet objectif a été rempli de façon satisfaisante.

Lors de nos différents apprentissages nous avons remarqué qu'il y avait peu de différences entre les joueurs. Les tables apprises sur différents joueurs étaient en effet semblables. Il semble que cela soit induit par la plateforme d'expérimentation, qui pousse les joueurs à adopter un comportement très agressif. L'utilisation de plateformes alternatives où la mort est pénalisée (Wolfenstein 3D, Counter-Strike, Soldier of Fortune, etc) pourrait permettre d'obtenir plus de diversité.

Nous avons abordé dans le premier chapitre, le critère d'humanité pour nos personnage synthétiques. Il paraît donc intéressant de se demander si ayant appris par imitation d'un être humain, les bots pourraient paraître plus humains, mais ce n'est pas le cas principalement à cause des états de base sur lesquels nous n'avons pas effectué d'apprentissage. En effet comme nous l'avons vu lors de la comparaison entre les états du bot et leur contrepartie humaine, certains états sont fort différents d'un comportement humain comme par exemple DetectionDanger.

Pour dépasser les limites de l'apprentissage que nous avons affectué, plusieurs pistes sont ouvertes. Tout d'abord, nous n'avons effectué l'apprentissage qu'au niveau de la sélection des actions, mais il serait très intéressant d'étendre la programmation bayésienne à un niveau plus bas et de pouvoir aussi effectuer l'apprentissage à ce niveau. On pourrait alors apprendre des caractéristiques du jeu d'un joueur et faire jouer un bot qui émulerait ses performances et son style (précision, armes préférées, utilisation des mouvements tactiques, etc). L'apprentissage serait alors relativement simple, toutes les variables étant bas niveau, nous n'avons besoin d'en déterminer aucune.

On pourrait également rajouter un niveau d'abstraction plus élevé. Le bot peut déjà charger les tables de probabilité dynamiquement. Lorsqu'on change de table, on change de comportement, on pourrait créer un modèle psychologique du joueur ([NC03]), et faire changer le bot de comportement suivant des variables affectant la psychologie du joueur (score, efficacité, etc). Mais effectuer un tel apprentissage serait difficile car il faudrait déterminer quel est le type de jeu du joueur (agressif, prudent, etc). Cela pose

directement la question du pouvoir discriminatoire du modèle : si j'ai calculé les tables d'un joueur, puis-je retrouver quel est ce joueur en comparant avec d'autres tables ?

Bibliographie

[AMS+01] R. Adobbati, A.N.Marshall, A.Scholer, S.Tejada, G.A.Kaminka, S.Schaffer and C.Sollitto. Gamebots : A 3D virtual world test-bed for multi-agent research. In second International Workshop on Infrastructure for agents, MAS, 2001.

[BDL+98a] P.Bessière, E. Dedieu, O. Lebeltel, E.Mazer and K. Mekhnacha. Interprétation vs. description i : Fondements mathématiques. Intellectica, pages 357-311, 1998.

[BDL+98b] P.Bessière, E. Dedieu, O. Lebeltel, E.Mazer and K. Mekhnacha. Interprétation vs. description ii : Proposition pour une théorie probabiliste des systèmes cognitifs sensori-moteurs. Intellectica, pages 313-336, 1998.

[CMK02] A.Cornujols, L.Miclet, Y.Kodratoff Apprentissage Artificiel. Edition Eyrolles, 2002

[Jay03] E.T.Jaynes. Probability theory - The logic of science. April 2003, Cambridge

[KT99] M.Kallmann, D.Thalmann. Direct3D interaction with smart objects. Proc. ACM VRST 99, London.

[KVS+02] G.A.Kamina, M.Veloso, S.Schaffer, C.Sollito, A.N.Marshall, R.Adobbati, S.Scholer, S.Tejata. Gamebots : the ever challenging multi-agent research test-bed. Communication of the ACM, January 2002.

[Lai01] J. E. Laird. Using a computer game to develop advanced AI. Computer, July 2001.

[Lai02] J. E. Laird. Research in Human-Level AI Using Computer Games. Communication of the ACM, January 2002.

[LD00] J. E. Laird and J. C. Duchi. Creating human-like synthetic characters with multiple skill-levels : A case study using the soar quakebot. In AAAI Fall Symposium Technical report, August 2000.

[Leb99] O. Lebeltel. Programmation Bayésienne des Robots. PhD thesis, Institut National Polytechnique de Grenoble, FRANCE, 1999.

[LeHy02] R. Le Hy. Programmation Bayésienne de Personnages de Jeux Vidéos. Mémoire de DEA, Institut National Polytechnique de Grenoble, FRANCE, 2002.

[LVL00] J. E. Laird and M. Van Lent. Human-level AI's Killer Application : Interactive Computer Games. AAAI, 2000.

[NC03] B.M.Namee and P.Cunningham. Enhancing non player characters in computer games using psychological models. ERCIM No 53, April 2003

[Rey87] C.Reynolds. Flocks, Herds, and Schools : A Distributed Behavioral Model. Computer Graphics vol. 21, 1987

[Woo01] S. Woodcock. Game AI : The state of the industry 2000-2001. Game Developer, August 2001

[KDW03] N.Kirby, E.Dysband, S.Woodcock Game Developer's Conference, AI Roundtable Report. GDC, March 2003

[Tur50] A. M. Turing. Computing Machinery and intelligence. Mind, 1950.