Programmation Bayésienne de Personnages de Jeux Vidéos

Ronan Le Hy

encadré par Pierre Bessière et Olivier Lebeltel

RAPPORT DE STAGE DE DEA

Juin 2002

Institut National Polytechnique de Grenoble
46, avenue Félix-Viallet
38031 Grenoble cedex 1

Ronan Le Hy (encadré par Pierre Bessière et Olivier Lebeltel)

Programmation Bayésienne de Personnages de Jeux Vidéos

Programmation Bayésienne de Personnages de Jeux Vidéos

Ronan Le Hy (encadré par Pierre Bessière et Olivier Lebeltel)

rlehy@free.fr

Remerciements

Je remercie en premier lieu mes maîtres de stage, Pierre Bessière et Olivier Lebeltel pour leur encadrement souple et éclairé, mais aussi pour m’avoir offert un sujet de stage qui fait un excellent sujet de conversation pour les dîners en ville.

Je remercie également David Raulo, thésard à l’INRIA, dont les discussions ont été extrêmement enrichissantes quant aux mondes virtuels et à l’approche bayésienne du contrôle de personnages synthétiques.

introduction

Ce stage de DEA Sciences Cognitives a été effectué à l'INRIA Rhône-Alpes du 1er Mars au 30 Juin 2002, au sein de l'équipe SHARP, sous la direction de Pierre Bessière et Olivier Lebeltel. Il porte sur la programmation bayésienne de personnages de jeux vidéos.

Nous explorons l'intérêt de l'application d'une technique de programmation des robots aux jeux vidéos. Nous espérons montrer les bénéfices de cette approche probabiliste en termes pratiques. Pour cela, nous utilisons une plateforme basée sur un jeu commercial nommé *Unreal Tournament* [1], pour lequel nous nous attachons à développer un joueur synthétique.

Nous exposons en premier lieu la problématique de l'intelligence artificielle dans les jeux vidéos, du point de vue de l'indutrie et de celui de la recherche ; cela nous conduit à énoncer les critères d'évaluation de notre méthode par rapport à celles existantes. Nous détaillons ensuite la construction de notre modèle bayésien, puis son ajustement sous deux formes : par spécification manuelle exhaustive, puis par apprentissage par l'exemple. Enfin, nous analysons les apports de notre méthode dans le domaine de la programmation de personnages dans les jeux vidéos. Cela nous conduit en particulier à une comparaison avec la technique classique des automates à états finis.

Une synthèse du cadre technique de notre étude est fournie en annexe. Cela inclut une description de la plateforme de jeu utilisée (*Unreal Tournament* et *Gamebots*), et du système de contrôle que nous avons développé par dessus pour acueillir notre schéma de programmation bayésienne.

[1] *Unreal Tournament* est une marque déposée de Epic Games, Inc.

Chapitre 1

problematique de l'IA dans les jeux vidéos

1.1 problématique industrielle

1.1.1 impératifs industriels

Nous dressons ici une liste de ce qui peut être vu comme des contraintes ou des objectifs lors du développement d'agents pour jeux vidéos. Une discussion plus détaillée sur ce sujet a été faite par LAIRD ([Lai00a]).

du point de vue du joueur

L'objectif central est la performance comportementale, qui doit s'approcher au mieux de celle d'un joueur humain. Cela implique des capacités de raisonnement spatial, de mémoire, de raisonnement de bon sens, d'établissement de buts, de tactique, de planification, de communication et de coordination, d'adaptation, d'imprévisibilité, et de personnalité.

Un point important qui vient contraindre cet objectif de performance est que le personnage synthétique ne doit pas tricher. Autant que possible, ses perceptions et possibilités d'action doivent êtres calquées sur celles du joueur humain. Dans le cas contraire, l'intérêt du jeu est fortement diminué.

du point de vue du programmeur

La première contrainte pour le programmeur est la productivité. Cela implique que pour lui la facilité de développement et la puissance expressive du mode de description sont essentielles.

Par ailleurs, pour qu'une méthode soit exploitable, elle doit conduire à des résultats prédictibles et contrôlables. Les comportements produits doivent être ajustables, pour éventuellement en produire différentes variantes. A cet effet, toutes les méthodes permettant un ajustement par apprentissage sont un avantage.

contraintes industrielles

Enfin, une méthode de programmation de personnage synthétique est utilisable si elle s'intègre bien dans le schéma de développement d'une application complexe, et si ses besoins de temps de calcul sont limités. En effet, l'industrie fait généralement une séparation claire entre les développeurs et les concepteurs de niveaux et de personnages ; ces derniers doivent pouvoir manipuler une interface simple sans

maîtriser les aspects techniques qui la sous-tendent. De plus, le temps processeur réservé à l'intelligence artificielle dans un jeu est typiquement de l'ordre de 10% à 20% ([Woo01b]).

1.1.2 techniques classiques employees

Nous faisons ici un survol rapide des techniques d'intelligence artificielle couramment utilisées dans l'industrie du jeu vidéo. Il faut noter l'absence de certaines méthodes bien implantées dans la recherche académique, comme les réseaux de neurones ou les algorithmes génétiques. Ces principes ont du mal à s'implanter dans les pratiques industrielles ([Woo01b]), principalement pour leur réputation – justifiée ou non – d'être peu prédictibles et difficiles à maîtriser.

systèmes de scripts

La méthode de programmation de personnages artificiels de loin la plus commune est l'écriture de scripts, dans un langage propre ou en C, sans méthodologie codifiée de haut niveau. Parmi les jeux utilisant ce mode de programmation, on trouve Unreal Tournament ([unr]), Civilization II ([civ], ou Baldur's Gate ([bal]).

La puissance expressive de ce type de méthode est bonne, dans la mesure où le langage de scripts utilisé est généralement propre au moteur de jeu. Cependant, ces langages étant souvent des langages impératifs ou objets classiques augmentés d'un système de gestion d'évènements, ils ne fournissent pas un cadre qui faciliterait particulièrement la programmation comportementale. Un script classique se ramène généralement à des cascades de `si ... alors ... sinon ...`. Ce type de systèmes impose donc aux programmes de conserver une taille limitée pour éviter de rendre les heuristiques ingérables par le programmeur.

Avec ces systèmes de programmation, la programmation et l'ajustement des comportements est réservée à des programmeurs, et ne peut être confiée à des scénaristes. Cependant, le temps de calcul est bien maîtrisé, dans la mesure où l'API de programmation est de suffisament haut niveau et où les scripts restent de taille raisonnable (pour la raison de complexité de conception évoquée plus haut).

Enfin, grâce à ce type de méthode, il suffit de rendre le moteur de script accessible pour permettre la modification du jeu par tous (ce qui est un argument commercial).

logique, moteurs d'inférence, systèmes experts

Des approches logiques sont faites de la programmation comportementale. Par exemple, John Funge présente un langage de type Prolog dédié à la programmation de comportements ([FTT99]) ; on peut aussi citer SOAR ([Lai00b]), architecture de contrôle proche d'un système expert.

Ces systèmes, faisant souvent l'objet de recherches académiques, ne s'imposent pas dans l'industrie. Les raisons de ce fait sont probablement leur lourdeur et leur gourmandise en temps de calcul. Malgré cela, ils autorisent une expressivité très grande, et intègrent bien certains problèmes d'apprentissage (pour SOAR : [Lai00b]).

flocking

Une technique particulièrement applicable aux jeux mettant en scène de nombreuses unités simples est celle du *flocking*. Cela consiste à essayer de faire émerger des comportements de groupe à partir de comportements élémentaires « sociaux ». [Woo01a] montre ainsi comment synthétiser un comportement

de troupeau à partir de quatre règles comportementales élémentaires qu'appliquent séparément chacun des individus contrôlés : séparation, alignement, cohésion et évitement.

Ce type de méthode donne des résultats souvent saisissants pour peu d'efforts de programmation. Cependant, le contrôle en est difficile, et on ne peut en envisager l'utilisation que dans le cas d'unités simples en grand nombre. Par ailleurs, les contraintes de temps de calcul et d'ajustabilité du comportement de masse produit font que les comportements élémentaires doivent rester simples.

automates

Un automate à états finis est un ensemble d'états reliés par des transitions étiquetées par des évènements ou des combinaisons de valeurs des variables décrivant le système. Chaque état correspond à un mode de fonctionnement du système, et les transitions d'état à état sont franchies selon les valeurs des variables internes. Un exemple d'implémentation pratique appliquée aux jeux vidéos de cette technique classique peut être trouvée dans un article d'E. Dysband ([Dys00]).

Ce mode de structuration des programmes a l'intérêt de fournir un schéma simple et bien compris de construction d'un comportement à partir du séquencement de plusieurs comportements élémentaires. Cependant, les transitions étant exprimées sous forme logique, ce type de structure se prête mal à l'apprentissage. Enfin, un automate nécessite très peu de calcul, et peut être implanté de manière très légère (voir par exemple [Car01]).

Les automates sont aussi utilisés dans leur version floue et non déterministe – dont un exemple pratique est donné dans un article d'E. Dysband ([Dys01]). Ils gagnent alors en expressivité, au prix d'une légère complication du formalisme. Cependant, ils sont sujets à l'explosion combinatoire du nombre de cas à traiter au niveau des transitions quand le nombre de variables augmente (une méthode pour tenter de réduire cette explosion est exposée dans [Zar01]).

Une comparaison des techniques d'automates déterministes à états finis et de notre méthode de programmation bayésienne sera faite au 5.2.

1.2 problématique de recherche

Les jeux vidéos fournissent aujourd'hui des problèmes mettant en jeu des interactions complexes avec le joueur humain. Les rôles qu'un personnage synthéthique peut jouer sont multiples. Il peut être ([LVL00]) ennemi tactique (comme dans la présente étude), partenaire du joueur humain, personnage secondaire de soutient du scenario, opposant stratégique, simple unité parmi d'autres, ou encore commentateur... Dans tous les cas l'objectif pratique est d'avoir un personnage qui agit comme agirait un joueur humain.

Devant cet objectif d'une intelligence artificielle de niveau humain, l'idée d'une forme de test de Turing ([Tur50]) survient : peut-on créer un joueur synthetique qui fasse illusion aux yeux d'un humain ? Cette approche est d'ailleurs suggérée par Laird ([LD00]), où une métode d'évaluation de personnages synthétiques est proposée ; un des critères pris en compte est l'« humanité ». Pour l'évaluer, des juges observent des joueurs humains et synthétiques en action, et tentent de déterminer lesquels sont humains, et lesquels sont artificiels. Cette idée met en valeur le fait que l'objectif recherché est bien une performance de niveau humain, et non une performance la meilleure possible ; par exemple, un délai de réaction trop court désignera un personnage comme artificiel, alors même qu'il représente un avantage du point de vue de l'efficacité au jeu.

Les jeux vidéos présentent les avantages pratiques d'autoriser un mode de développement non monolothique sur des plateformes très variées et peu onéreuses, et d'avoir le soutient d'une industrie puissante.

Celle-ci a des besoins en termes de résultats comme en termes de méthodes, et possède des moyens financiers, ainsi qu'une forte expertise technologique dans les champs connexes (animation, interfaces...).

Finalement, les jeux vidéos apparaissent comme « un environnement riche pour le développement incrémental d'une IA de niveau humain » ([LVL00]).

1.3 cadre pratique

Nous avons fait le choix de travailler sur un jeu de combat à la première personne : *Unreal Tournament*, agrémenté d'une modification nommée *Gamebots* ([AMS+01], [KVS+02], [gam]) permettant de s'y connecter facilement. Ce jeu met en scène des personnages humanoïdes se déplaçant dans un monde tridimensionnel. Le joueur contrôle un de ces joueurs de l'intérieur. Le but est de tuer un maximum de fois les autres joueurs, à l'aide d'armes, de munitions, et de bonus de santé ramassés dans le niveau. Un comportement performant dans cet environnement a donc une composante de dextérité (viser juste, sauter et courir au bon endroit...), et une autre de gestion de ressources (ramasser les armes les plus fortes, savoir fuire quand son niveau de vie est bas...).

Les modalités techniques de notre étude ainsi que l'architecture de contrôle choisie sont détaillées en annexe. Le fait que le type de jeu choisi soit bien adapté aux comportements réactifs, et l'accès simple au moteur de jeu fourni par *Gamebots* sont les principaux avantages de cette plateforme.

Dans la suite de ce rapport, nous utilisons le terme *bot* (prononcé comme *botte*) pour désigner ces robots virtuels supposés agir comme des humains. Nos *bots* sont des personnages d'un jeu de combat, dont l'apparence n'est pas différente de celle des avatars commandés par des joueurs humains, et qui tendent au même but, à savoir toucher ses ennemis un nombre maximum de fois.

1.4 méthodologie d'évaluation

Compte-tenu des objectifs exposés au 1.4 et de la plateforme technique choisie, nous avons les objectifs pratiques suivants :

- mettre en place un schéma de programmation bayésienne pour le comportements de personnages, et l'évaluer en regard des méthodes existantes ;
- mettre en oeuvre et évaluer une méthode d'ajustement du comportement programmé par aprentissage.

Il faut noter que ces objectifs n'engagent pas à construire un personnage qui soit absolument efficace au combat ; il est question d'élaborer une méthode et d'en montrer les avantages pratiques.

Le problème posé est d'évaluer les comportements programmés au regard des critères énoncés au 1.4, que l'on peut résumer de la façon suivante :

1. performance du bot :
   (a) score
   (b) « humanité » : raisonnement spatial, mémoire, bon sens, buts, tactique, planification, communication et coordination, adaptation, imprévisibilité, personnalité
2. développement :
   (a) facilité de développement
   (b) puissance expressive
3. comportements :

   (a) prévisibilité
   (b) ajustabilité
4. contraintes industrielles :
   (a) séparation développement / conception des personnages
   (b) temps de calcul nécessaire limité

Une solution pratique pour évaluer le critère 1a (score) est de laisser les personnages programmés se comparer au combat à des bots existants. Nous avons essayé cette solution, en utilisant comme référence un bot développé sur la même plateforme (*Gamebots*). Cependant, les résultats de ce côté ne sont pas aussi etoffés qu'on le voudrait à cause de l'instabilité des bots de référence disponibles. Nous avons donc aussi comparé les différentes versions de nos bots entre elles.

Le critère 1b (« humanité ») met en jeu des aspects purement délibératifs et communicationnels, que nous avons délibérément laissés de côté dans cette étude. Nous retiendrons donc essentiellement les aspects d'imprévisibilité et de personnalité. Nous retiendrons aussi l'aspect stratégique plutôt que tactique, puisque c'est à ce niveau que se place notre modèle comportemental. Enfin, Nous laissons de côté le problème de l'adaptation en ligne, pour nous focaliser sur l'apprentissage hors ligne[1]. Les notions retenues sont qualitatives et subjectives ; nous tenterons cependant de donner des éléments d'appréciation tangibles et termes de comportement.

Les critères 2a (facilité de développement) et 2b (puissance expressive) seront évalués au regard du temps pris pour programmer un comportement, et de la variété des comportements possibles au sein de notre modèle.

Nous évaluerons les criteres 3a (prévisibilité) et 3b (ajustabilité) au regard de la facilité de programmer différents comportements avec certaines caractéristiques particulières (comme l'agressivité, et la prise en compte du niveau de santé).

La question 4a (séparation développement/conception) se verra abordée sur la base de l'organisation de notre modèle ; 4b (complexité) à partir du calcul mis en jeu dans par modèle.

[1] C'est-à-dire où les phases d'apprentissage et de restitution sont séparées.

Chapitre 2

modèle de programmation bayésienne de personnages

2.1 programmation bayésienne des robots

Nous faisons ici un très bref rappel du principe de la programmation bayésienne des robots ([BDL+98a], [BDL+98b],[Leb99]). Nous ne prétendons pas à l'exhaustivité, mais à jeter les bases de son application aux jeux vidéos.

2.1.1 organisation générale

Un programme bayésien (voir figure 2.1) est composé d'une description, et d'une question. La description est constituée du choix des variables pertinentes pour le problème, de la décomposition de la distribution conjointe, et de l'écriture des formes paramétriques.

La question consiste à interroger la distribution en faisant un tirage sur une distribution de probabilité.

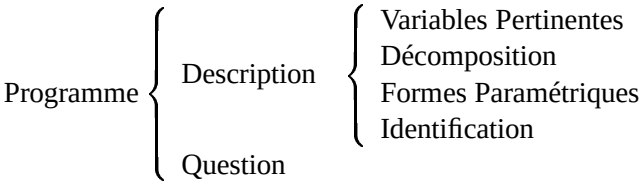

FIG. 2.1 – structure d'un programme bayésien

Plutôt que de réiterer les considérations formelles des références données plus haut, nous explicitons la méthode sur un exmple simple (sans rapport avec notre bot).

2.1.2 exemple

Considérons un personnage très simple pouvant se trouver dans deux états : attaque et fuite. Ce personnage dispose de son niveau de vie et (faible ou fort) de son arme (faible ou forte). A partir de ces informations, il doit décider à chaque instant dans quel état passer.

variables pertinentes

Les variables pertinentes sont :
– l'état $E \in attaque, fuite$
– le niveau de vie $V \in faible, fort$
– l'arme $A \in faible, forte$

décomposition

Nous choisisson d'écrire la décomposition comme :

$$P(E\ A\ V) = P(E)P(V|E)P(A|E)$$

L'hypothèse d'indépendance conditionnelle nécessaire à l'écriture de la décomposition sous cette forme est que A et V sont indépendants conditionnellement à E.

formes paramétriques

Nous choisissons les formes paramétriques suivantes :
– $P(E)$ est uniforme ;
– $P(V|E)$ et $P(A|E)$ sont définies dans les tables 2.1 et 2.2.

	$attaque$	$fuite$
$faible$	0	x
$fort$	x	0.1

TAB. 2.1 – $P(V|E)$

	$attaque$	$fuite$
$faible$	0.1	x
$forte$	x	x

TAB. 2.2 – $P(A|E)$

Le format de ces tables est le suivant. La première colonne fait la liste des valeurs de la variable cherchée, la première ligne celle des valeurs des la variable connue. Par exemple, pour la table 2.1 la case $(1, 2)$ correspond à $P(V = faible|E = fuite)$. Les cases marquées d'un x sont calculées de façon à de ce que chaque colonne soit normalisée, c'est-à-dire $\forall j, \sum_i P(Cherché = Cherché_i|Connu = Connu_j) = 1$.

identification

Toutes les formes élémentaires étant complètement spécifiées, nous n'avons pas d'identification à faire. L'identification pourrait par exemple consister à apprendre les tables au lieu de les écrire complètement.

question

Notre question est : $P(E|A\ V)$. Elle se résout de la façon suivante :

$$P(E|A\ V) = \frac{P(E\ A\ V)}{P(A\ V)} = \frac{P(E)P(V|E)P(A|E))}{\sum_E P(E)P(V|E)P(A|E)}$$

Une décision est ensuite prise à partir de la distribution résultante. Elle peut consister à choisir la valeur de E de plus grande probabilité, ou à effectuer un tirage.

2.2 description

2.2.1 variables pertinentes

Les variables choisies représentent les espaces moteurs et sensoriels du bot au sein du modèle. En particulier, les variables sensorielles font la représentation du monde du bot : ce sont celles qui lui permettent de caractériser son état interne et son environnement en temps réel, et de prendre des décisions.

Pour faire le choix de ces variables, nous avons pris en compte les informations disponibles au sein de notre plateforme. Nous en avons gardé un certain nombre d'informations simples et vraisemblablement pertinentes pour un humain qui jouerait. Enfin, nous avons cherché à conserver un nombre réduit de variables pour garder le modèle relativement simple.

Les variables choisies le sont aussi dans la perspective d'un comportement réactif. Cela exclut par exemple des caractéristiques délibératives comme la planification.

variables d'état

Ces variables sont l'équivalent des variables motrices en robotique.

$E_t \in \{A, RA, RV, Ex, Fu, DD\}$: l'état du bot à l'instant t (les valeurs possibles sont détaillées plus bas).

$E_{t+1} \in \{A, RA, RV, Ex, Fu, DD\}$: l'état du bot à l'instant $t+1$.

$Tir \in \{Faux, Vrai\}$: drapeau indiquant au bot de faire feu.

Les valeurs possibles des variables d'état E_t et E_{t+1} sont :

A **:** (attaque) le bot recherche un ennemi s'il n'en a pas à proximité, et l'attaque.

RA **:** (recherche arme) le bot part à la recherche d'une arme.

RV **:** (recherche vie) le bot part à la recherche d'un bonus de santé.

Ex **:** (exploration) le bot explore les parties du niveau non encore visitées.

Fu **:** (fuite) le bot cherche à s'éloigner des ennemis en présence.

DD **:** (détection danger) le bot observe autour de lui pour détecter un éventuel danger.

variables sensorielles

$Vie \in \{Bas, Moyen, Haut\}$: le niveau de vie du bot.

$Arme \in \{Aucune, Moyenne, Forte\}$: la force de l'arme du bot.

$ArmeAdversaire \in \{Aucune, Moyenne, Forte\}$: la force de l'arme de l'adversaire s'il y en a un.

$Bruit \in \{Faux, Vrai\}$: drapeau indiquant si un bruit a été entendu récemment.

$NombreEnnemis \in \{Aucun, Un, DeuxOuPlus\}$: le nombre d'ennemis proches.

$ProxArme \in \{Faux, Vrai\}$: drapeau indiquant la proximité d'une arme.

$ProxSante \in \{Faux, Vrai\}$: drapeau indiquant la proximité d'un bonus de santé.

2.2.2 décomposition

La distribution conjointe est décomposée de la manière suivante :

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

Deux hypothèses d'indépendance conditionnelle sont formulées pour écrire cette décomposition. La première est que les variables perceptives (Vie, $Arme$, $ArmeAdversaire$, $Bruit$, $NombreEnnemis$, $ProxArme$, $ProxSante$) sont indépendantes deux à deux sachant E_{t+1}.

La seconde hypothèse est que sachant $NombreEnnemis$, Tir ne dépend pas des autres variables. Cela revient à faire le choix que Tir ne variera que selon $NombreEnnemis$.

Cette décomposition met en jeu une la probabilité du passage d'un état à l'autre ($P(E_{t+1}|E_t)$), celle du tir en fonction du nombre d'ennemis proches, ainsi que celle de chaque variable perceptive sachant l'état suivant le moment où la variable est mesurée ($P(...|E_{t+1})$). Nous allons donc spécifier et utiliser la distribution de chaque variable perceptive en supposant connu l'état à l'instant suivant. Cette démarche, appelée *programmation inverse*, peu paraître peu naturelle au premier abord ; cependant nous verrons qu'elle rend l'identification par écriture à la main ou par apprentissage aisée et efficace.

2.2.3 formes paramétriques

- $P(E_t)$ est choisi uniforme.
- Toutes les autres distributions sont spécifiées dans des tables. La partie suivante (3) est centrée sur l'écriture et la génération de ceux-ci.

2.2.4 identification

L'identification consiste ici à fixer les valeurs des tables de probabilités. Elle peut être faite manuellement, ou par apprentissage,

manuelle

Une forme d'identification manuelle peut être faite. Comme sur notre exemple-jouet du 2.1.2, les tables sont écrites manuellement.

apprentissage

L'apprentissage que nous envisageons ici est supervisé : un humain pilote le bot à partir d'une interface lui permettant de sélectionner en temps réel l'état du bot et de suivre son évolution dans le jeu (voir figure 2.2).

FIG. 2.2 – interface de pilotage du bot

Cet apprentissage permet de fixer les tables de probabilités sous la forme de lois de succession de Laplace. Cette loi, paramétrée par les n_{ij}, s'écrit :

$$\forall V, \forall i, \forall j, P(V = V_i | W = W_j) = \frac{1 + n_{ij}}{\sum_i n_{ij} + |V|}$$

où V et W sont des variables, n_{ij} est le nombre d'occurences du cas $(V = V_i, W = W_j)$, et $|V|$ est le nombre de valeurs possibles pour V.

Il est intéressant de remarquer que loi de succession de Laplace part, pour un nombre nul d'échantillons, d'un *a priori* uniforme sur la distribution :

$$\forall (i,j), n_{ij} = 0 \Rightarrow P(V = V_i | W = W_j) = \frac{1}{|V|}$$

De plus, quand le nombre de d'échantillons devient grand, la loi tend vers celle donnée par les fréquences :

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

Enfin, il est notable que cette loi n'assigne jamais de valeur de probabilité nulle. Une discussion plus poussée son l'intérêt peut être trouvée dans un *Probability Theory : the Logic of Science*, d'E. T. Jaynes ([Jay]).

2.3 question

La première question posée au modèle pour faire agir le bot est : « quelle est la distribution de l'état à l'instant suivant, sachant l'état à l'instant courant et les valeurs des variables sensorielles ? » ; soit, dans notre formalisme :

$$P(E_{t+1}|E_t\, Vie\, Arme\, ArmeAdversaire\, Bruit\, NombreEnnemis\, ProxArme\, ProxSante)$$

Elle est résolue de la façon suivante :

$$\begin{aligned}
&P(E_{t+1}|E_t\ Vie\ Arme\ ArmeAdversaire\ Bruit\ NombreEnnemis\ ProxArme\ ProxSante)\\
&= \frac{P(E_{t+1}\ E_t Vie\ Arme\ ArmeAdversaire\ Bruit\ NombreEnnemis\ ProxArme\ ProxSante)}{P(E_t\ Vie\ Arme\ ArmeAdversaire\ Bruit\ NombreEnnemis ProxArme\ ProxSante)}\\
&= \frac{1}{Z} \times (P(E_t)P(E_{t+1}|E_t)P(Vie|E_{t+1})P(Arme|E_{t+1})P(ArmeAdversaire|E_{t+1})P(Bruit|E_{t+1})\\
&\quad P(NombreEnnemis|E_{t+1})P(ProxArme|E_{t+1})P(ProxSante|E_{t+1}))\\
&= \frac{1}{Z'} \times (P(E_{t+1}|E_t)P(Vie|E_{t+1})P(Arme|E_{t+1})P(ArmeAdversaire|E_{t+1})P(Bruit|E_{t+1}) \quad (2.1)\\
&\quad P(NombreEnnemis|E_{t+1})P(ProxArme|E_{t+1})P(ProxSante|E_{t+1}))
\end{aligned}$$

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

On remarque que $P(E_t)$ n'intervient pas dans la résolution de la question.

La seconde question porte sur le Tir : $P(Tir|NombreEnnemis)$. Elle est spécifiée directement dans le modèle et ne requiert pas d'inférence.

La résolution de la première question met en valeur l'influence de chaque distribution élémentaire sur la distribution de l'état à l'instant suivant. En particulier, si une probabilité $P([S = S_i]|[E_{t+1} = E_i])$ vaut zéro, cela implique que $P([E_{t+1} = E_i]|...[S = S_i]...)$ est nulle aussi. Nous voyons donc là le moyen d'interdire de passer dans un état pour une valeur donnée d'une variable sensorielle.

Chapitre 3

programmation d'un comportement

Nous identifions ici nos formes paramétriques par écriture directe. Nous nous attachons à montrer, à travers les commentaires de toutes les tables utilisées, de quelle façon ces tables sont structurées et comment elles influencent le comportement synthétisé.

3.1 tables de comportement

3.1.1 format des tables

Chaque table est donnée en deux versions, qui correspondent à deux types de comportements : *odge*, bot ordinaire, et *brsk* (berserk), bot très agressif.

3.1.2 $P(E_{t+1}|E_t)$

La table 3.1 correspond aux probabilités de transitions entre états. Elle est construite pour stabiliser les états (*auto-maintien*). Les probabilités de transition entre deux états différents sont fixées à une valeur faible.

La version *brsk* présente une particularité : outre les x de la diagonale correspondant à la force de l'auto-maintien, la probabilité est toujours grande de basculer dans l'état *Attaque*.

3.1.3 $P(Vie|E_{t+1})$

La table 3.2 montre comment mettre en relation une variable et l'état du bot. Par exemple, sachant que le bot bascule dans l'état *Recherche de Vie*, la probabilité qu'il ait un niveau de vie élevé est de $0,001$, celle que son niveau de vie soit moyen de $0,01$, et celle que son niveau de vie soit faible, $1 - 0,01 - 0,001 = 0989$.

Le comportement *brsk* est indifférent à la variable Vie. Il consiste en effet à attaquer aveuglément, sans se préocupper de son niveau de vie.

3.1.4 $P(Bruit|E_{t+1})$

La table 3.3 est identique pour les deux comportements envisagés. Elle spécifie en particulier que le bot a peu de chances de percevoir un bruit dans les états de recherche de santé ou d'arme, d'exploration ou de détection de danger.

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. 3.1 – $P(E_{t+1}|E_t)$

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. 3.2 – $P(Vie|E_{t+1})$

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. 3.3 – $P(Bruit|E_{t+1})$

3.1.5 $P(NombreEnnemis|E_{t+1})$

La table 3.4 détaille la probabilité du nombre d'ennemis selon l'état du bot. On cherche ici à spécifier que *odge* a tendance à fuire s'il est en présence de deux ennemis ou plus, alors que le bot *brsk* est porté à attaquer quand les ennemis sont nombreux. Les deux bots explorent quand il n'ont pas d'ennemis. Cette connaissance est exprimée sous forme inverse : nous écrivons donc par exemple que sachant que le bot *brsk* passe dans l'état Attaque, il a de grandes chances $(0,8)$ d'avoir deux ennemis ou plus, de petites chances (0.2) d'avoir un ennemi, et aucune chance de pas avoir d'ennemi. Cela lui interdit donc effectivement de passer en attaque quand il n'a pas d'ennemi et son excitation accrue quand de nombreux ennemis sont présents. Pour le bot *odge*, on peut noter que sachant que l'état suivant est la recherche d'armes, les chances sont infimes que des ennemis soient présents.

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. 3.4 – $P(NombreEnnemis|E_{t+1})$

3.1.6 $P(ProxArme|E_{t+1})$ et $P(ProxSante|E_{t+1})$

Les tables 3.5 et 3.6 rendent le bot opportuniste quand il passe à côté d'une arme ou d'un bonus de santé : certains états sont exclus quand un objet est proche. En particulier, la probabilité d'être proche d'une arme quand on bascule dans les états RV, Ex ou DD est faible.

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. 3.5 – $P(ProxArme|E_{t+1})$

3.1.7 $P(Arme|E_{t+1})$

La table 3.7 montre que l'arme courante du bot a une influence interdit ou influence le basculement vers certains états (A, RA, Fu), mais ne joue pas sur le basculement vers d'autres (RV, Ex, DD).

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. 3.6 – $P(ProxSante|E_{t+1})$

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. 3.7 – $P(Arme|E_{t+1})$

3.1.8 $P(ArmeAdversaire|E_{t+1})$

La table 3.8 spécifie que le comportement d'attaque est inhibé par l'arme de l'adversaire si elle est forte, particuliérement pour le comportement *odge*.

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. 3.8 – $P(ArmeAdversaire|E_{t+1})$

3.1.9 $P(Tir|NombreEnnemis)$

La table 3.9, identique pour les deux comportements envisagés, décrit une heuristique de tir simple : le bot fait feu s'il a des ennemis autour de lui. Il faut noter que cette table simple est quasiment équivalente à l'utilisation d'un simple

$$Tir \leftarrow (NombreEnnemis \neq Aucun)$$

L'avantage de la gestion bayésienne de la variable Tir est l'homogénéité du modèle.

brsk	$Aucun$	Un	$DeuxOuPlus$
$Faux$	x	0.001	0.001
$Vrai$	0.001	x	x

odge	$Aucun$	Un	$DeuxOuPlus$
$Faux$	x	0.001	0.001
$Vrai$	0.001	x	x

TAB. 3.9 – $P(Tir|NombreEnnemis)$

3.2 résultats

Dans ces résultats, nous faisons référence aux critères d'évaluation établis au 1.4.

1a score

La version *brsk* se montre plus efficace et prend généralement une avance au score de 20% à 30%. On observe que dans l'état de la plateforme sur laquelle tourne notre modèle, la fuite semble un mauvais choix, souvent plus dangeureux que l'attaque.

1b « humanité »

Ce critère est relié à celui de prévisibilité. Les bots se comportent, vus de l'intérieur comme de l'extérieur (c'est-à-dire en voyant ou non leur état interne) de manière plausible ; dit autrement, ils ne donnent pas l'impression d'agir au hasard.

2a facilité de développement

L'écriture de ces tables est relativement systématique. Elle a été faite en une vingtaine de minutes. On peut cependant arguer que celle-ci deviendrait impraticable si le nombre de variables sensorielles ou celui des états possibles augmentait grandement. Dans le cas de l'augmentation du nombre des valeurs possibles des variables, il serait sans doute plus intéressant d'utiliser des formes paramétriques fonctionnelles. Par ailleurs, la démarche de programmation inverse est particulière et parfois peu intuitive ; cependant, l'habitude permet de maîtriser cette technique qui est centrale à notre modèle.

2b puissance expressive

Les limites de la puissance expressive de notre modèle sont dans le choix de nos variables. En effet, nous contrôlons avec ces tables l'influence de chacune des variables sensorielles sur l'état du bot. Nous avons vu en particulier comment interdire ou inhiber à différents niveaux un état quand une variable prend une valeur donnée.

3a prévisibilité

Les deux comportements programmés ici montrent après essais leur plausibilité. Il apparaît en effet que les bots changent d'état de manière sinon toujours prévisible, au moins compréhensible. Les tables de probabilités remplissent leur rôle selon les attentes.

3b ajustabilité

Il est intéressant de constater que les différences de comportement que nous avons cherché à introduire entre les versions *odge* et *brsk* apparaissent dans le jeu. Elle se montre comme prévu plus agressive,

et ses différentes particularités recherchées se retrouvent. Par exemple, elle ne se préoccuppe pas de son niveau de vie, et bascule très rarement dans l'état Fuite.

4a séparation développement/conception

Pour implanter les comportements, le seul travail nécessaire a été de saisir les tables dans des fichiers. Notre modèle est en effet adapté au chargement des comportements à la volée, sans recompilation ni modification des routines de calcul. Il est donc clair que la séparation entre l'implantation du modèle et son ajustement peuvent être complètement séparés.

4b complexité

L'écriture des tables ne met pas en jeu de calcul. La question de la complexité est donc ici sans fondement.

Chapitre 4

apprentissage d'un comportement

Notre propos ici est d'ajuster un comportement, les connaissances préalables étant posées. On cherche donc à déterminer les tables à partir d'un apprentissage.

Il s'agit donc d'ajuster un modèle réactif à un comportement humain complexe. Cela exclut de reproduire ce comportement dans sa totalité, en particulier dans ses aspects de planification. De plus, notre bot ne possède pas de représentation structurée de l'espace, ce qui écarte sans discussion qu'il puisse prendre en compte les même variables que le joueur humain. Cependant, nous avons l'espoir de pouvoir ajuster un modèle réactif imitant certains des aspects du jeu, à savoir la gestion des ressources (arme, santé) et des objectifs (atteindre son adversaire sans être atteint).

4.1 pilotage par comportement

Le pilotage par comportement, déjà évoqué au 2.2.4, consiste à fournir au concepteur de personnage une interface lui permettant de piloter le bot. Cette interface (figure 2.2) comporte d'une part la fenêtre de jeu d'*Unreal Tournament* centrée sur le bot, d'autre part une fenêtre de contrôle de l'état interne du bot. Cette fenêtre inclut des boutons qui permettent de sélectionner en temps réel le comportement du bot.

Les résultats que nous présentons ont été collectés avec un apprentissage d'une durée de l'ordre de 10 à 15 mn. Cela correspond à environ 7000 échantillons servant à paramétrer les lois de succession de Laplace utilisée pour l'apprentissage.

4.2 tables

Nous commentons ici les tables[1] apprises, par comparaison avec celles écrites au 3.1. On note que bien qu'il soit possible d'apprendre la table concernant $P(EtZ)$, cela n'a pas d'intérêt dans la mesure où cette distribution se trouve simplifiée dans le calcul des questions posées au modèle (voir la formule 2.1). Par ailleurs, la table P($Tir\ |NombreEnnemis$) n'est pas apprise, pour la difficulté que représenterait le fait de contrôler deux valeurs simultanément, et pour celle d'évaluer en temps réel quels sont les ennemis que le bot perçoit, compte-tenu de son angle de vision réduit.

[1]Pour une lecture plus aisée, les tables sont données arrondies, avec deux chiffres significatifs. Les probabilités données comme 1 sont donc en réalité comprises entre 0.995 et 1.

4.2.1 $P(E_{t+1}|E_t)$

La table 4.1 est très proche, dans l'esprit que celle correspondante (3.1) écrite à la main (version *odge*). La seule différence notable est la valeur d'auto-maintien de l'état Fu, qui est plus faible $(0,29)$ ici. Cela s'explique par le fait que le pilotage d'un bot à la main ne conduit pas souvent à fuire ; ainsi, $P(E_{t+1}|[E_t = Fu])$ s'éloigne peu de l'uniforme par manque de données. Pour cette raison, dans la suite de ce commentaire, nous ne tiendrons pas compte du comportement Fu.

appris	A	RA	RV	Ex	Fu	DD
A	1	0.00055	0.0024	0.001	0.14	0.0054
RA	$8.4e-06$	1	0.0024	0.001	0.14	0.0054
RV	$8.4e-06$	0.00055	0.99	0.001	0.14	0.0054
Ex	$8.4e-06$	0.00055	0.0024	0.99	0.14	0.0054
Fu	$8.4e-06$	0.00055	0.0024	0.001	0.29	0.0054
DD	$8.4e-06$	0.00055	0.0024	0.001	0.14	0.97

TAB. 4.1 – $P(E_{t+1}|E_t)$

4.2.2 $P(Vie|E_{t+1})$

La table apprise (4.2) est ici assez différente de celles écrites à la main (3.2). Cela est dû au fait qu'il est difficile lors du pilotage en temps réel de tenir compte en continu du niveau de vie ; il est souvent plus naturel d'être opportuniste en ramassant les objets proches, et en attaquant de manière suicidaire sinon. Chaque colonne de la table apprise reflète donc grossièrement la probabilité du niveau de vie du bot, indépendamment de l'état.

appris	A	RA	RV	Ex	Fu	DD
Bas	0.11	0.017	0.0024	0.0041	0.5	0.038
$Moyen$	0.89	0.74	0.8	0.57	0.25	0.96
$Haut$	0.00027	0.24	0.2	0.43	0.25	0.0055

TAB. 4.2 – $P(Vie|E_{t+1})$

4.2.3 $P(Bruit|E_{t+1})$

La table 4.2 montre que pour le comportement appris, le bruit est indifférent à la plupart des comportements, sauf à Ex et RV, qui sont quasi-incompatibles avec le bruit (ces comportements s'interrompent quand un ennemi attaque, par exemple).

appris	A	RA	RV	Ex	Fu	DD
$Faux$	0.37	0.58	0.91	0.81	0.67	0.55
$Vrai$	0.63	0.42	0.093	0.19	0.33	0.45

TAB. 4.3 – $P(Bruit|E_{t+1})$

4.2.4 $P(NombreEnnemis|E_{t+1})$

La table 4.4 est identique à celle spécifiée à la main (version *odge*) pour les comportements RA, Ex et DD. La colonne RV est similaire à RA, contrairement à la spécification à la main qui la voyait uniforme ; cela s'explique par un comportement piloté qui prone l'attaque quand un ennemi est présent. Enfin, dans la table apprise, l'attaque exclut la présence de deux ennemis ou plus, comme la version *odge*.

appris	A	RA	RV	Ex	Fu	DD
$Aucun$	0.62	0.84	0.96	0.94	0.5	0.85
Un	0.38	0.14	0.041	0.061	0.25	0.13
$DeuxOuPlus$	$8.4e-06$	0.018	0.0024	0.001	0.25	0.022

TAB. 4.4 – $P(NombreEnnemis|E_{t+1})$

4.2.5 $P(ProxArme|E_{t+1})$ et $P(ProxSante|E_{t+1})$

Les tables 4.5 et 4.6 correspondent grossièrement à celles écrites à la main ; cependant, les proportions sont beaucoup moins marquées. Il faut ici remarquer pour expliquer ce point que le pilotage permet difficilement d'évaluer en temps réel quels objets sont perçus comme proches par le bot.

appris	A	RA	RV	Ex	Fu	DD
$Faux$	0.49	0.58	0.89	0.38	0.67	0.57
$Vrai$	0.51	0.42	0.11	0.62	0.33	0.43

TAB. 4.5 – $P(ProxArme|E_{t+1})$

appris	A	RA	RV	Ex	Fu	DD
$Faux$	0.97	0.82	0.49	0.75	0.67	0.81
$Vrai$	0.025	0.18	0.51	0.25	0.33	0.19

TAB. 4.6 – $P(ProxSante|E_{t+1})$

4.2.6 $P(Arme|E_{t+1})$

La table 4.7 a peu de rapports avec celle spécifiée à la main. Elle reflète principalement le fait que le bot est la quasi-totalité du temps avec une arme de classe $Moyenne$. Cela conduit à penser que la classification des armes devrait être remaniée, ou que le comportement de recherche d'armes n'est pas assez efficace.

4.2.7 $P(ArmeAdversaire|E_{t+1})$

La table 4.8 donne plus d'informations que celle écrite à la main (3.8). Elle est caractéristique du fait que quand aucun adversaire n'est présent, la variable $ArmeAdversaire$ vaut $Aucune$. Par conséquent, on voit ici que dans les comportements RA, RV et Ex, la probabilité qu'un ennemi soit présent est très faible. Par ailleurs on retrouve le fait que la probabilité de l'arme de l'ennemi sachant que le bot

appris	A	RA	RV	Ex	Fu	DD
$Aucune$	$1.7e-05$	0.00056	0.0024	0.001	0.5	0.0055
$Moyenne$	0.81	0.95	1	0.99	0.25	0.76
$Forte$	0.19	0.054	0.0024	0.012	0.25	0.24

TAB. 4.7 – $P(Arme|E_{t+1})$

est en attaque exclut pratiquement que l'ennemi ait une arme très forte. En somme, la table issue de l'apprentissage suggère une autre façon d'exprimer certaines notions que la spécification à la main ; cette autre façon est adaptée aux spécificités de l'architecture de contrôle.

appris	A	RA	RV	Ex	Fu	DD
$Aucune$	0.62	0.84	0.96	0.94	0.5	0.85
$Moyenne$	0.38	0.086	0.041	0.05	0.25	0.1
$Forte$	0.0032	0.069	0.0024	0.012	0.25	0.049

TAB. 4.8 – $P(ArmeAdversaire|E_{t+1})$

4.3 résultats

1a score

Au combat, le bot issu de l'apprentissage se montre moins efficace que ceux programmés à la main (des confrontations à trois entre les versions apprise, *odge* et *brsk* conduisent à des scores[2] moyens de l'ordre de 2, 5 et 7 respectivement).

1b « humanité »

Il se pose notamment un problème d'opportunisme : la version apprise du bot profite mal de la présence proche d'armes et de bonus de santé pour les ramasser. Des raisons pour expliquer ce problème, ainsi que celui plus général de l'efficacité seront proposées au 4.3.

2a facilité de développement

L'ajustement d'un comportement par apprentissage représente une méthode idéale de développement. Les résultats sur les tables indiquent que le pilotage par comportement conduit à des résultats acceptables. Cependant celui-ci empêche le joueur de faire jouer son expertise, qui est associée à l'interface propre du jeu.

2b puissance expressive

La puissance expressive du modèle n'est pas affectée par l'apprentissage, en ce qu'elle dépend principalement de la description utilisée, et en particulier du choix des variables.

[2]Le score correspond au nombre de joueurs tués.

3a prévisibilité

Dans l'ensemble, nous retrouvons dans les tables apprises les principes comportementaux injectés dans les tables écrites à la main. Il y a cependant un certain nombre de différences notables. Ces différences ne sont pas des problèmes en soi (l'objectif n'étant pas de faire apprendre un comportement identique à celui spécifié à la main), mais sont seulement caractéristiques de différences entre les comportements appris et spécifiés *a priori*. Elles deviennent problématiques quand les distributions obtenues sont indûment[3] proches de l'uniforme.

En effet, une distribution proche de l'uniforme contient très peu d'information ; les variables associées en partie gauche sont neutralisées (ici, par exemple $P(Bruit|[E_{t+1} = DD])$ pour le comportement appris). Ce phénomène est caractéristique d'une inadéquation entre les variables prises en compte par le pilote humain, et celles contenues dans le modèle (comme les problèmes du bruit et de la proximité des objets).

3b ajustabilité

L'apprentissage permet, dans la limite des capacités du modèle, de programmer une variété de comportements. Cependant, on peut arguer que le contrôle sur certains aspects précis est moindre que lors de l'écriture manuelle de toutes les tables. En effet, si certaines situations importantes ne se reproduisent pas souvent lors de l'apprentissage, les probabilités qui les conditionnent seront mal estimées.

4a séparation développement/conception

L'apprentissage peut être vu comme le niveau ultime de la séparation entre développement et conception de personnages. En effet, la personne ajustant le comportement n'a plus alors besoin d'aucune connaissance du modèle sous-jacent. Il est possible, par exemple, de profiter de l'expertise d'un joueur ne prenant pas part au développement.

4b complexité

Du point de la complexité, notre méthode d'apprentissage requiert de stocker autant de fréquences qu'il y a de valeurs de probabilité dans les tables, et de mettre à jour une valeur dans chacune des tables à chaque pas de temps. L'opération est donc légère en temps de calcul, et les besoins en mémoire évoluent quadratiquement en fonction du nombre d'états et linéairement en fonction du nombre de variables sensorielles :

$$Memoire \leq |E|^2 + |E| \times \max_i(|S_i|)$$

La solution aux problèmes de performances liés à cette méthode d'apprentissage consisterait soit à retravailler et à étoffer le modèle pour l'approcher de celui pris en compte par le joueur humain, soit à fournir à celui-ci une interface qui limite sa perception à celle que connait le bot. Cependant, d'un côté nous n'avons pas accès au modèle du joueur (à supposer même qu'il puisse être exprimé dans le cadre de la programmation bayésienne) ; de l'autre, réduire l'interface de pilotage au maximum rendrait celui-ci impraticable, puisqu'elle retirerait au pilote son expertise, qui est justement associée à l'environnement « naturel » du jeu. Nous proposerons cependant des pistes pour tenter de rapprocher l'homme et le bot au 5.3.

[3] C'est-à-dire alors qu'elles devraient clairement être marquées pour que le comportement soit efficace.

Chapitre 5

discussion

5.1 intérêts et inconvénients de la méthode

Nous reprenons ici les critères d'évaluation tels que nous les avons énoncés au 1.4, de façon à mettre en avant les avantages et les failles de la méthode de programmation bayésienne appliquée aux personnages de jeux vidéos.

1a score

Il est important de remarquer que comme celle de tout modèle de haut niveau, l'efficacité pratique de notre modèle dépend de architecture de contrôle sur laquelle il est basé, et d'une plateforme fournissant les variables nécessaires. C'est la raison pour laquelle il est difficile de comparer nos bots à ceux existants pour *Unreal Tournament*, dans la mesure où ceux-ci ont un accès plus direct à leur monde, alors que les notres en ont une vision « humanisée », qui limite leur perception et leurs déplacements. Si elle est faite, cette comparaison est cruelle, tant l'avantage de dextérité perceptive (par exemple dans l'identification des dangers) des bots natifs d'*Unreal Tournament* est grand. Par ailleurs, confrontés à des bots programmés sur la même plateforme (*Gamebots*), nos bots se montrent à la hauteur.

Nous ne pouvons donc conclure sur ce point particulier, puisqu'il mesure beaucoup plus que l'efficacité de notre modèle bayésien. C'est l'efficacité de toute notre plateforme (*Unreal Tournament*, *Gamebots*, système de contrôle, modèle bayésien) qui influe ce résultat. Nous pensons cependant qu'il est possible, en couplant notre modèle à une plateforme performante d'obtenir un ensemble compétitif, grâce aux avantages sur les autres critères, dont la description suit.

1b « humanité »

Notre modèle porte spécifiquement sur la programmation d'un comportement stratégique. Nous avons montré que cette démarche était couronnée de succès : les décisions prises transcrivent bien la stratégie telle qu'elle a été conçue. Notre modèle conduit à une évaluation des situations et à des prises de décisions conformes aux attentes.

Pour ce qui est de l'imprévisibilite, il semble clair que l'utilisation d'un schéma de décision non déterministe comme le nôtre est un avantage. Alors que les décisions sont conformes aux attentes, elles ne sont pas prévisibles individuellement avec précision.

Nous avons montré que notre méthode permettait, au sein d'une description fixée, de donner une

personnalité propre à un bot en ajustant de manière sélective certaines de ses caractéristiques comportementales.

Enfin,nous avons d'emblée écarté les aspects délibératifs de notre étude, mais il semble indispensable de les prendre en compte tôt ou tard. En particulier, les travaux de David Raulo ([RAC00]) se dirigent sur la voie de l'intégration des problèmes de navigation et de localisation au modèle bayésien.

2a facilité de développement

Le formalisme de la programmation bayésienne des robots est clairement défini et fournit une manière simple et concise de décrire un programme. Il est bien adapté au développement incrémental. D'une part, il est simple d'ajouter une variable sensorielle V à la description : cela revient à spécifier une ligne et une colonne de plus dans la matrice de transition $P(E_{t+1}|E_t)$, ainsi que de spécifier la forme paramétrique $P(V|E_{t+1})$. D'autre part, les schèmas de programmation comprenant une description et une question sont réutilisables comme formes élémentaires au sein d'une description de plus haut niveau.

Cependant, dans l'état de la technique, l'interfaçage du système de programmation bayésienne avec les systèmes perceptifs classiques dans les jeux vidéos reste plus problématique que celui réalisable avec un système de script. Le problème est au niveau du traitement de variables à valeurs discrètes (arme) ou composées (tir ou bruit : un booléen et une direction) : cela complique l'utilisation de fonctionnelles en lieu de formes paramétriques, et pousse à celle de tables dont tous les éléments doivent être écrites à la main.

2b puissance expressive

La programmation avec notre méthode est du même niveau que les variables choisies dans la description. On a donc un bon contrôle sur la puissance expressive. Elle peut même s'étaler à plusieurs niveaux. Ainsi, sur notre exemple, nous pourrions envisager d'écrire un schéma pour le comportement d'attaque, ainsi (au niveau inférieur) que pour le comportement de déplacement élémentaire.

3a prévisibilité

Il est important pour le programmeur de savoir quels seront les répercussions sur le comportement de ses choix, tant dans le choix des variables et de la distribution conjointe qu'au niveau de l'identification des formes paramétriques. Nous pensons avoir montré avec les comportements *odge* et *brsk* que les attentes exprimés lors de l'écriture du modèle sont satisfaites dans sa réalisation.

3b ajustabilité

Les comportements construits a partir de la programmation bayésienne sont spécifiés par atomes contrôlables et ajustables séparément (les formes paramétriques). L'influence de chaque variable sensorielle sur l'état du bot y est spécifiée. Ainsi, dans notre démarche, nous avons pu contrôler separement l'influence du bruit sur l'etat du bot (voir table 3.3). L'influence de la variation des formes paramétriques est observable et compréhensible, comme nous l'avons montré avec les comportements *odge* et *brsk*. Enfin, les formes paramétriques sont facilement apprenables, qu'elles soient des tables comme dans cette étude ou des fonctionnelles (comme dans [Leb99]).

L'apprentissage s'intègre naturellement à notre modèle, et offre une façon très puissante de spécifier un comportement de manière intuitive et accessible

Quant à l'apprentissage, s'il s'intègre naturellement à notre modélisation bayésienne, sa forme actuelle de pilotage par comportement semble insuffisante. En effet, elle n'est pas naturelle pour le pilote.

4a séparation développement/conception

Pour de mêmes connaissances préalables, différents comportements sont décrits par des données (les formes paramétriques), et non du code. Cela implique une flexibilité dans le developpement de comportements multiples, et dans leur utilisation (sauvegarde et chargement au vol, par exemple).

Par ailleurs, une fois le modèle écrit, il peut etre ajusté séparément, à la main ou par apprentissage. Celui-ci offre une façon très puissante de spécifier un comportement sans avoir de connaissance approfondie du modèle utilisé.

4b complexité

Du point de vue de la complexité, notre méthode se montre extrêmement économe. Si nous notons $|E_{t+1}|$ le nombre de valeurs possibles pour E_{t+1} et n le nombre de variables sensorielles, la formule 2.1 montre que l'établissement de la probabilité $P(E_{t+1}|E_t\, Vie Arme...)$ met en jeu :

- $|E_{t+1}| \times (n+1)$ lectures dans des tables,
- $|E_{t+1}| \times n$ multiplications,
- $|E_{t+1}| - 1$ additions, et
- $|E_{t+1}|$ divisions.

La complexité du calcul est donc linéaire dans le nombre de variables sensorielles. En revanche, elle ne dépend pas du nombre de cas de ces variables, mais seulement linéairement du nombre d'états possibles. A titre de comparaison, dans un automate déterministe à états finis, le passage d'un état à un autre nécessite au plus $|E_{t+1}| \times (n+1)$ comparaisons permettant d'identifier la situation sensorielle à une transition. Il est donc clair que notre méthode est très performante en termes de temps de calcul, et le reste quand le nombre de variables sensorielles et la granularité de celles-ci augmentent au-delà de notre exemple-jouet.

5.2 comparaison avec un automate équivalent

Au tirage sur les distributions près[1], notre modèle en action se comporte comme un automate dont les états sont les valeurs de la variable E_t et dont les transitions portent des conditions logiques sur l'ensemble des variables perceptives ; cet automate a un nombre d'état réduit mais des transitions très complexes.

Il est possible de dresser une table de transitions donnant l'état le plus probable à partir d'un état et d'une configuration de variables sensorielles donnés.

A partir d'un état donné, les transitions partitionnent l'espace sensoriel de cardinal

$$|Vie| \times |Arme| \times |ArmeAdversaire| \times |Bruit||NombreEnnemis| \times |ProxArme| \times |ProxSante| = 648$$

en $|E_{t+1}| = 6$ ensembles.

Ces transitions sont donc des formules logiques du type :

$(E_t = A \wedge Vie = Bas \wedge Arme = Aucune \wedge ArmeAdversaire = Aucune \wedge ((Bruit = Faux \wedge NombreEnnemis \neq Aucun) \vee (Bruit = Vrai \wedge ((NombreEnnemis = Un \wedge ProxArme = Vrai) \vee NombreEnnemis = DeuxOuPlus)))) \vee ... \Rightarrow E_{t+1} = Fu$

[1] Si nous remplaçons notre tirage par le choix de la valeur de plus grande probabilité, cette réserve est levée.

Il est clair que l'écriture à la main de toutes ces formules est pratiquement infaisable. Elles doivent prendre en compte toute la logique des transitions entre états, sont longues et difficiles à lire et à modifier. Pour que cette méthode reste utilisable, il est nécessaire de limiter le nombre de variables sensorielles ainsi que le nombre de cas de ces variables, dans la mesure où le nombre de cas à prendre en compte correspond à la taille de l'espace sensoriel. Par ailleurs, ces règles sont difficilement apprenables dans le cadre du formalisme logique classique.

Par comparaison, notre méthode autorise la définition de l'automate et de toutes ses transitions de manière légère, contrôlable et maintenable, à partir de briques élémentaires (les formes paramétriques, ici sous forme de tables).

5.3 perspectives

Pour dépasser les limites de la méthode de programmation bayésienne pour les jeux vidéos telle que nous l'avons présentée ici, plusieurs pistes sont ouvertes. La première est de descendre plus bas dans la hiérarchie des comportements implémentés, pour permettre par exemple la programmation bayésienne et l'apprentissage des comportements de base (attaque, fuite, …). Cela étendrait les avantages de la programmation bayésienne à une part plus importante de la programmation du bot, qui repose encore sur une méthode de programmation classique, et autoriserait l'amélioration des performances de dextérité perceptive.

Une seconde piste est l'utilisation de formes paramétriques fonctionnelles, qui pourraient permettre des spécifications plus aisées, et encore plus concises que celles sous forme de tables. Cela permettrait de prende une compte une meilleure granularité des variables perceptives.

Un seconde piste est l'exploration de nouvelles formes d'apprentissage auxquelles le schéma bayésien semble approprié. Il serait souhaitable de pouvoir piloter le bot directement à travers l'interface native du jeu ; cela permettrait un véritable « transsfert de compétences » du joueur au bot, sous réserve que le modèle de départ soit approprié. Cependant, il faut noter que ce type d'aprentissage soulève le problème de la reconnaissance en temps réel du comportement suivi. Par ailleurs, un apprentissage non supervisé par renforcement est envisageable.

Enfin, il serait intéressant de se pencher sur d'autres types de jeux que ceux de combat à la première personne. Ceux-ci ont une forte composante de dextérité, qui restreint l'avantage qu'une méthode donnée peut proposer au niveau stratégique. Certain jeux de rôle feraient sans doute une plateforme mieux adaptée à la mise en évidence d'avantages en termes d'efficacité d'une méthodologie de haut niveau comme la nôtre.

conclusion

En guise de conclusion, nous voulons soulever la question de l'incertitude. En effet, la programmation bayésienne des robots que nous mettons en application ici prétend « traiter formellement des problèmes d'incertitude et d'incomplétude inhérents au domaine considéré » ([Leb99]). Or il peut sembler que notre domaine d'étude, les mondes virtuels, ne présente pas la même incertitude motrice et perceptive propre aux robots réels : les informations de position ou de vitesse sont connues précisément, et les déplacements et les actions sur le monde ont des résultats prévisibles.

Mais la place de l'incertain est double dans les mondes virtuels. Premièrement, la complexité des mondes virtuels est aujourd'hui suffisante pour que les personnages y évoluant, humains ou virtuels, ne puissent en avoir qu'une vision incomplète. Deuxièmement, l'incertitude est d'abord dans la tête du programmeur. Nous ne savons formaliser nos raisonnements sur le monde ordinaire (c'est-à-dire hors des mathématiques) qu'avec difficulté. Nous avons vu au 5.2 combien il peut être difficile d'exprimer en termes logiques la totalité des cas qui peuvent faire passer de l'attaque à la fuite un personnage dont l'état est caractérisé par seulement 6 variables à 2 ou 3 états chacune. En dehors du fait que le formalisme que nous exposons réduise grandement la quantité d'informations à fournir par le programmeur pour synthétiser son raisonnement, il lui permet d'exprimer celui-ci sous une forme intuitive.

Nous pensons avoir montré que la programmation bayésienne est bien adaptée au champ des mondes virtuels. Elle possède des avantages sur les méthodes classiques, même si certains de ces avantages (comme l'apprentissage) restent à concrétiser sous une forme satisfaisante.

Annexe A

cadre technique de l'etude

A.1 plateforme

A.1.1 *Unreal Tournament* et *Gamebots*

Notre plateforme déxpérimentation est constitué d'un jeu de combat à la première personne, *Unreal Tournament* ([unr]), augmenté d'une modification intitulée *Gamebots* ([AMS+01], [KVS+02]). Cette modification permet le contrôle de bots à distance par le réseau. Elle fournit un protocole de communication qui permet à un client contrôlant un joueur virtuel de recevoir des informations perceptives locales au joueur, et d'envoyer des ordres moteurs. Par ailleurs, il offre un service de recherche de chemin entre deux points. Un des intérêts de cette plateforme est de fournir une interface de contrôle « humanisée » : les bots contrôlés par *Gamebots* ne sont pas omniscients et disposent d'informations perceptives et d'ordres moteurs limités. L'inconvénient inhérent à cette approche est qu'elle introduit d'entrée un sévère handicap des bots contrôlés *via Gamebots* par rapports aux bots natifs d'*Unreal Tournament*.

A.1.2 API de contrôle des bots

Nous donnons ici une liste simplifiée des informations perceptives et des ordres moteurs offerts par *Gamebots*. Les détails de cette API peuvent être trouvés en ligne sur [gam].

informations perceptives

personnage

- id, nom, équipe
- position (rotation, lieu), vitesse
- santé, armure, niveau de vie
- arme, munitions
- ramassé objet
- pieds, tête ou corps changent de zone (eau, lave...)
- changement d'arme (automatique ou provoqué)
- collision avec un mur, un objet ou un joueur
- chute
- mort, blessure

autres personnages visibles
- id, nom, équipe
- position (rotation, lieu), vitesse, accessibilité
- arme, fait feu
- mort, blessure infligée par soi

environnement dans le champ de vision
- nœuds de navigation : id, lieu, accessibilité
- portes, ascenseurs : id, lieu, accessibilité, type
- objets : id, lieu, accessibilité, type
- bruit (pas, ascenseur, tirs, objet ramassé)
- projectile se dirigeant vers soi
- réponse à une requête de chemin ou d'accessibilité

jeu
- scores, capture de drapeau
- message d'un autre joueur (texte ou typé)

ordres moteurs et requêtes

déplacement
- marcher, courir vers un point, un joueur, un objet, un nœud de navigation...
- courir vers un point en faisant face à un point/objet (strafe)
- se tourner vers un point/objet ou d'un angle
- s'arrêter
- sauter

armes
- armes
- commencer de tirer
- arrêter de tirer
- changer d'arme

requêtes
- chemin vers un point/objet
- accessibilité d'un point/objet
- message aux autres joueurs

A.2 architecture de contrôle

Notre architecture de contrôle est présentée sur la figure A.1. Elle est composée de trois processus légers (*thread*s) (mise à jour, décision, interface) agissant sur les données internes du bot, et communiquant avec la surcouche *Gamebots* d'*Unreal Tournament via* une socket. Le *thread*de mise à jour analyse les messages envoyés par *Gamebots*, et met à jour en conséquence les données internes du bot (les flèches indiquent le sens de circulation des données). Il déclenche aussi des évènements qui sont passés au

*thread*de décision par un tunnel de communication. Le *thread*de décision agit en envoyant des ordres à *Gamebots* à partir de la consultation des données internes. Enfin, une interface graphique affiche une partie des données internes à fins de pilotage et de contrôle de l'état du bot.

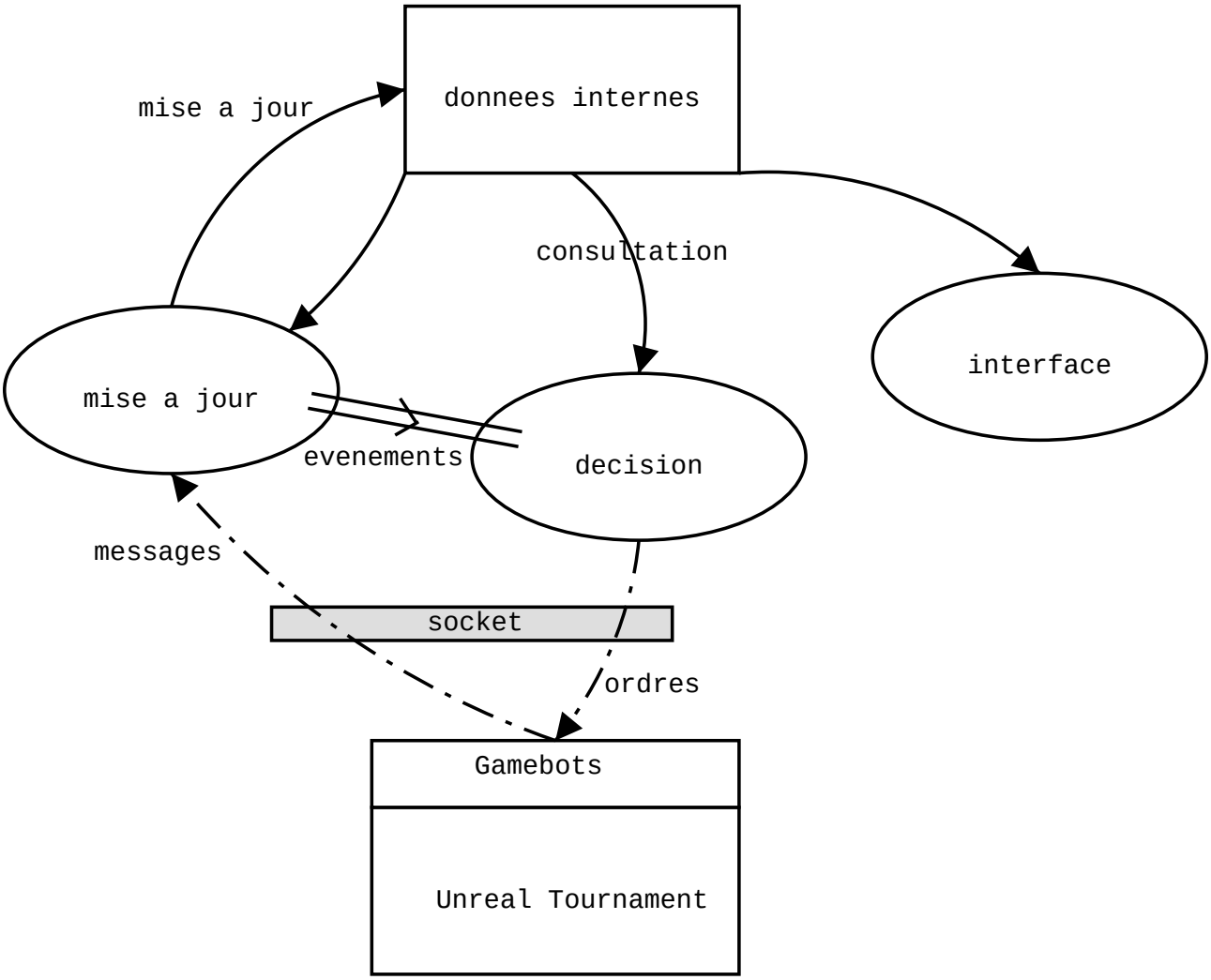

FIG. A.1 – architecture de contrôle

Nous illustrons la gestion du flux de contrôle par la figure A.2. Cette figure représente un gros plan sur le *thread*de décision. Au plus haut niveau, une décision est prise, et le comportement correspondant est lancé (`exploration`). Celui-ci appelle un comportement élémentaire (`aller_a`). Ce comportement de base cherche à se synchroniser sur l'arrivée d'un évènement depuis le *thread*de mise à jour (`attendre NOEUD_ATTEINT`). Si cet évènement est reçu (flèches vertes), `aller_a` continue à agir, jusqu' à retourner le contrôle à `explore_local`, qui le retourne enfin à la boucle de décision. Mais le *thread*de décision peut envoyer un évènement signalant le changement d'une des variables sensorielles à partir desquelles se fait la décision. Dans ce cas, à la réception du message dans `aller_a`, une exception est lancée et propagée (en rouge) jusqu'a la boucle de décision. Dans les deux cas, une nouvelle décision est prise, et le cycle recommence.

La figure A.3 montre la hiérarchie d'héritage des comportements élémentaires (pointillés). Les com-

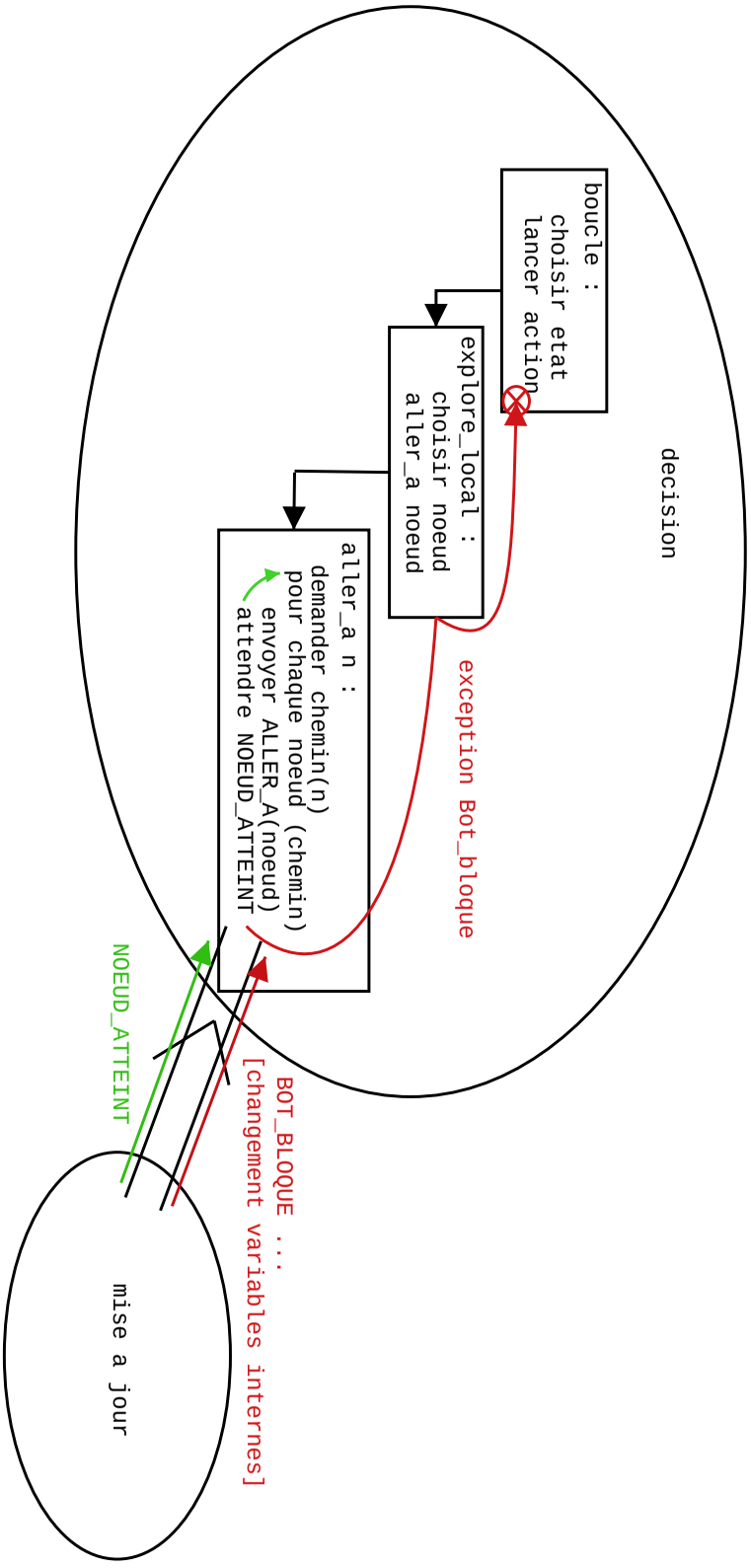

FIG. A.2 – flux de contrôle

portements séquencés par la boucle de décision bayésienne (`RechercheSante`, `RechercheArme`, `Fuite`, `Exploration`, `DetectionDanger`, `Attaque`) sont les feuilles de l'arbre. Tous les comportements implémentent un comportement abstrait spécifiant leur interface. Celle-ci consiste en une méthode permettant de reprendre un comportement à zéro, et une autre permettant de le continuer là où il a été arrêté. Les lignes pleines reliant `AllerA` à `RechercheObjet`, `Attaque` et `DetectionDanger` matérialisent le fait que ces trois comportements contiennent un comportement `AllerA`, qui gère au plus bas niveau le déplacement d'un point à un autre de la carte.

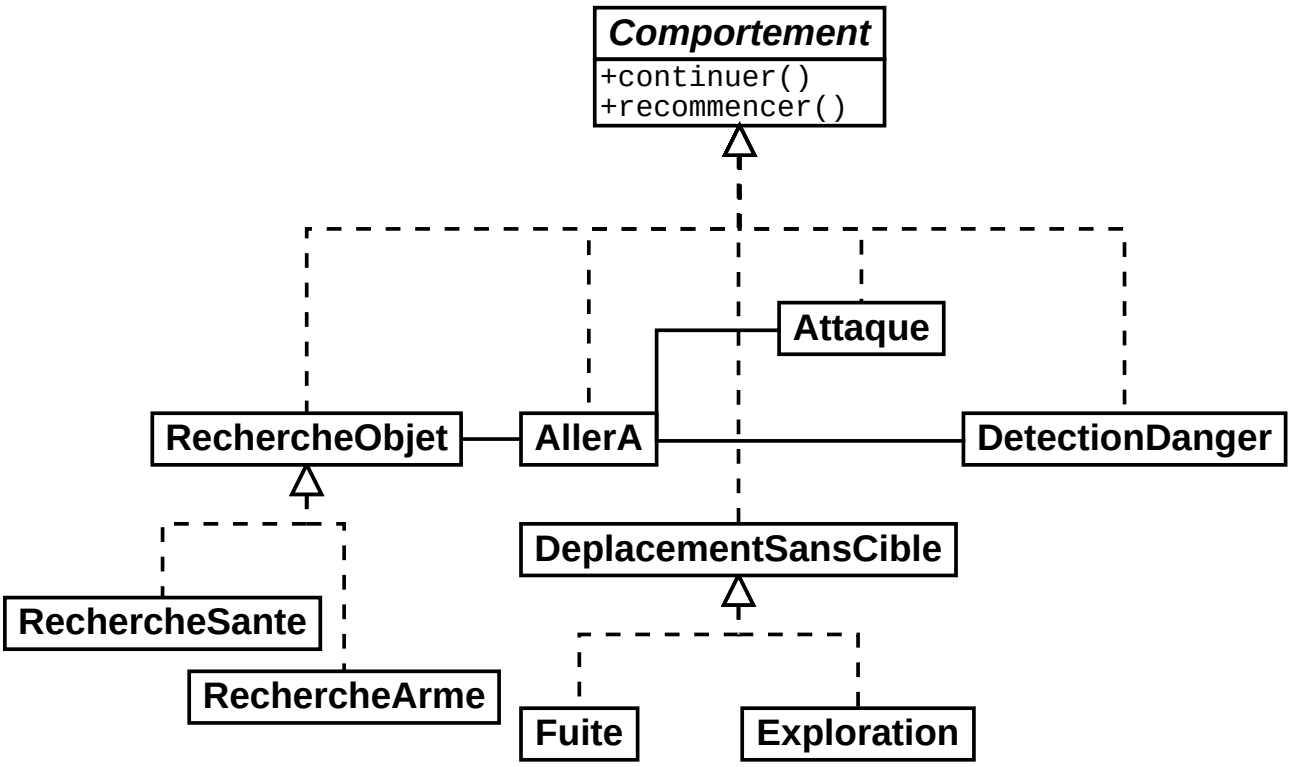

FIG. A.3 – comportements

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, and Scalable MAS*, 2001.

[bal] Baldur's gate scripting. Site web. http ://www25.brinkster.com/iesdp/.

[BDL+98a] P. Bessière, E. Dedieu, O. Lebeltel, E. Mazer, and K. Mekhnacha. Interprétation vs. description i : Proposition pour une théorie probabiliste des systèmes cognitifs sensori-moteurs. *Intellectica*, pages 257–311, 1998.

[BDL+98b] P. Bessière, E. Dedieu, O. Lebeltel, E. Mazer, and K. Mekhnacha. Interprétation vs. description ii : Fondements mathématiques. *Intellectica*, pages 313–336, 1998.

[Car01] S. Carter. Micro-threads for game object ai. In M. Deloura, editor, *Game Programming Gems 2*, pages 265–272. Charles River Media, 2001.

[civ] Slic : Civilization ii scripting. Site web. http ://apolyton.net/ctp2/modification/activision/slic.shtml.

[Dys00] E. Dysband. A finite-state machine class. In M. Deloura, editor, *Game Programming Gems*, pages 237–248. Charles River Media, 2000.

[Dys01] E. Dysband. A generic fuzzy state machine in c++. In M. Deloura, editor, *Game Programming Gems 2*, pages 337–341. Charles River Media, 2001.

[FTT99] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling : Knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH*, August 1999.

[gam] Gamebots api. Site web. www.planetunreal.com/gamebots.

[Jay] E.T. Jaynes. *Probability theory - The logic of science*. à paraître. Version provisoire disponible à http ://bayes.wustl.edu.

[KVS+02] G. A. Kaminka, M. Veloso, S. Schaffer, C. Sollitto, Andrew N. Adobbati, R.and Marshal, S. Scholer, Andrew, and S. Tejada. Gamebots : the ever-challenging multi-agent research test-bed. *Communications of the ACM*, January 2002.

[Lai00a] J. Laird. Design goals for autonomous synthetic characters. Draft., 2000.

[Lai00b] J. E. Laird. It knows what you're going to do : Adding anticipation to a quakebot. In *AAAI Spring Symposium Technical Report*, March 2000.

[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.

[LVL00] J. E. Laird and M. Van Lent. Human-level ai's killer application : Interactive computer games. In *AAAI Fall Symposium Technical Report*, August 2000.

[RAC00] D. Raulo, J.-M. Ahuactzin, and Laugier C. Controlling virtual autonomous entities in dynamic environments using an appropriate sense-plan-control paradigm. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Takamatsu (JP)*, November 2000.

[Tur50] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236) :433–460, 1950.

[unr] Unreal tournament. Site web. http ://www.unrealtournament.com.

[Woo01a] S. Woodcock. Flocking with teeth : Predators and prey. In M. Deloura, editor, *Game Programming Gems 2*, pages 330–336. Charles River Media, 2001.

[Woo01b] S. Woodcock. Game ai : The state of the industry 2000-2001. *Game Developer*, August 2001.

[Zar01] M. Zarozinski. Imploding combinatorial explosion in a fuzzy system. In M. Deloura, editor, *Game Programming Gems 2*, pages 342–350. Charles River Media, 2001.

Sommaire

Programmation Bayésienne de Personnages de Jeux Vidéos

Ronan Le Hy
(encadré par Pierre Bessière et Olivier Lebeltel)

Résumé

Nous présentons une méthode de spécification de comportement pour personnages de jeux vidéos.

Institut National Polytechnique de Grenoble
46, avenue Félix-Viallet
38031 Grenoble cedex 1