



Modélisation, Suivi et Apprentissage de Comportements Non-Procéduraux dans un Environnement Virtuel de Formation

Matthieu AUBRY

Professeur encadrant : Pierre DE LOOR

CERV : Centre Européen de Réalité Virtuelle
EA-3883 – LISYC

Laboratoire d'Informatique pour les SYstèmes Complexes
Rapport de stage de Master 2 Recherche Informatique IFSIC

Le 20 juin 2006

Remerciements

Je souhaite tout d'abord remercier les personnes qui m'ont permis de suivre la formation de Master 2 Recherche Informatique de l'IFSIC. En particulier M. François BODIN, responsable du Master, M. Djamil SARNI, responsable de la filière brestoise du Master, et pour finir M. Pierre DE LOOR responsable des étudiants de l'ENIB.

Je souhaite également remercier M. Jacques TISSEAU, directeur du Centre Européen de Réalité Virtuelle (CERV) pour nous avoir accueilli dans son laboratoire.

Je tiens aussi à remercier une nouvelle fois mon maître de stage M. Pierre DE LOOR pour sa disponibilité, son aide et les discussions que nous avons eu.

Je voudrais aussi adresser mes remerciements à M. Romain BENARD qui n'a jamais hésité à me consacrer du temps et qui a bien voulu partager le fruit de son travail avec moi.

Pour finir, je remercie Fabrice HARROUËT, pour les outils qu'il a mis à notre disposition, tous les élèves de Master, avec qui j'ai passé de bons moments, Tim, pour nous avoir diverti lors de quelques pauses déjeuner, et toutes les personnes qui m'ont soutenu.

Table des matières

1	Introduction	4
2	Bibliographie	6
2.1	Programmation Par Démonstration et IHM	7
2.1.1	Généralités	7
2.1.2	La création du plan	8
2.1.3	L'utilisation du plan	8
2.2	Nature des plans générés	9
2.2.1	Plans déterministes	9
2.2.2	Plans non-déterministes	10
2.3	Rapidité d'apprentissage	14
2.3.1	Les résultats obtenus	14
2.3.2	Les critères	14
2.4	Généralisation	15
2.4.1	Généralisation de paramètres	15
2.4.2	Structures de contrôles	15
2.5	Restitution	17
2.5.1	Adaptation à l'utilisateur	17
2.5.2	Assistance	17
2.5.3	Automatisation de tâches	17
2.6	Adaptation du plan	19
2.6.1	Modifications manuelles	19
2.6.2	Statistiques et préférences	19
2.6.3	Changement de tâche	20
3	Un modèle pour les comportements dynamiques	21
3.1	La notion de contexte	21
3.1.1	Définition	21
3.1.2	La notion de perception	22
3.1.3	Génération d'un contexte	23
3.1.4	Comparaison de deux contextes	23
3.2	Les graphes contextuels	23
3.2.1	Les différents types de noeuds	24
3.2.2	Les liens entre les noeuds	24
3.2.3	Graphe contextuel	25
4	Utilisation du modèle	27
4.1	Mécanismes génériques	27
4.1.1	Organisation des graphes	27
4.1.2	Parcours et interprétation de graphe	28
4.1.3	Fonctionnement d'un Pointer	29
4.2	Prise de décision	32
4.2.1	Extraction des informations	32
4.2.2	Fonctionnement avancé	32
4.3	Assistance pédagogique	33
4.3.1	Spécialisation	33
4.3.2	Deux types d'aides	33
4.3.3	Fonctionnement avancé	34

5	Apprentissage rapide de comportements	36
5.1	Données pour l'apprentissage	36
5.2	Filtrage des scénarios	37
5.2.1	Filtrage des contextes	38
5.2.2	Filtrage des perceptions	38
5.2.3	Reconnaissance d'actions cycliques	39
5.2.4	Reconnaissance de sous-graphes	39
5.3	Recherche des situations clés	39
5.3.1	Alignement des scénarios	39
5.3.2	Identification de points clés	40
5.4	Génération du graphe	41
6	Résultats	42
6.1	Implémentation	42
6.1.1	Gestion des DCxG	42
6.1.2	Gestion des pointeurs	43
6.2	Utilisation dans un exercice de foot	43
6.2.1	Exécution de graphes	44
6.2.2	Assistance pédagogique	45
6.3	Unification et mémoire	46
6.4	Collaboration	47
6.5	Apprentissage	48
6.5.1	Limites de l'alignement	48
6.5.2	Un exemple complet	48
7	Conclusion	50

Chapitre 1

Introduction

Le CERV¹, composante de l'équipe d'accueil EA-3883, rassemblant l'ENIB² et l'UBO³, mène des recherches dans le domaine de la réalité virtuelle. L'équipe SPI⁴ se consacre aux travaux sur les Environnements Virtuels de Formation (EVF).

Les EVF permettent à un utilisateur, ou apprenant, l'acquisition d'un savoir faire au travers d'un exercice virtuel. Afin de passer le cap de la simple simulation, un EVF pourra contenir un *système de tutorat intelligent* (ITS⁵). L'ITS devra alors guider l'apprenant dans sa démarche.

Pour fonctionner, un ITS va nécessiter un certain nombre de modèles :

- Le modèle du domaine va permettre de représenter les connaissances de l'expert, les règles absolues sur l'exercice.
- Le modèle d'interface abstrait les actions de l'utilisateur.
- Le modèle pédagogique permet au *tuteur* de savoir quand, pourquoi et comment intervenir.
- Le modèle de l'apprenant représente les capacités, le comportement et les connaissances de l'utilisateur.

Notre but est de contribuer à la construction du modèle de l'apprenant en représentant son comportement et en devinant ses actions par l'expression de son comportement.

L'étude bibliographique, proposée au chapitre 2, offre un tour d'horizon des différentes techniques utilisées en Programmation Par Démonstration (PBD⁶). La création et l'utilisation d'un modèle comportemental de l'utilisateur est un des buts des systèmes de PBD.

Nous n'étudierons cependant que l'utilisation de la PBD dans les IHM *intelligentes*. Cette restriction vient d'une analogie naturelle entre *Utilisateur/ Apprenant* et *Action sur l'interface/ Interactions avec le monde virtuel* (cf. FIG. 1.1).

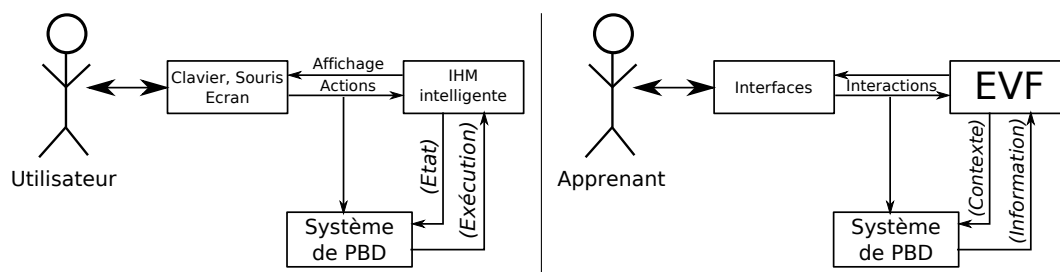


FIG. 1.1 – Analogie entre une IHM *intelligente* (à gauche) et un EVF (à droite)

Après une présentation de la PBD dans le domaine des IHM *intelligentes*, section 2.1, nous

¹Centre Européen de Réalité Virtuelle

²Ecole Nationale d'Ingénieurs de Brest

³Université de Bretagne Occidentale

⁴Simulation Participative et Immersive

⁵Intelligent Tutoring System

⁶Programming By Demonstration

étudierons les différents modèles proposés, section 2.2.

Nous attacherons ensuite une attention particulière à deux critères nous semblant important dans l'intégration de la PBD dans un EVF :

- Le système de PBD doit pouvoir créer les modèles comportementaux à partir d'un petit nombre d'itérations de l'exercice. En effet un apprenant ne voudra certainement pas reproduire des centaines de fois un exercice.
- La complexité des plans que l'on va pouvoir générer nous a aussi semblé primordiale. Est-ce que l'on va pouvoir abstraire une procédure, inclure des structures de contrôle ?

Dans les sections suivantes, nous étudierons alors les possibilités d'utilisation de ces plans. Pour cela, nous étudierons ce que peuvent restituer les systèmes de PBD et leurs capacités d'adaptation à l'utilisateur :

- Quelles sont les informations que nous allons pouvoir extraire des plans ?
- Est-il possible d'adapter un plan à plusieurs utilisateurs différents ?

Suite à cette étude, nous proposerons un modèle permettant la modélisation de comportement non-procéduraux dans un environnement dynamique. Basé sur les travaux réalisés en PBD, notre modèle prendra aussi en compte des spécificités propres aux environnements dynamiques.

La section 3.1 présente la notion de contexte, déjà utilisée dans [3] pour représenter l'état de l'apprenant dans une simulation dynamique et collaborative.

Dans la section suivante, nous présenterons le modèle lui-même, créé à partir de la notion de contexte. Nous verrons alors les contraintes imposées par un environnement dynamique et nos besoins, ainsi que la solution que nous avons proposé : les graphes contextuels dynamiques.

Le chapitre 4 propose un ensemble de mécanismes permettant l'exploitation des informations contenues par le modèle proposé précédemment.

La section 4.1 présente un mécanisme générique pour l'utilisation de notre modèle. Basé sur le parcours en temps réel de graphes, nous verrons comment l'appliquer à deux utilisations différentes. Nous verrons alors comment nous avons pu mettre en valeur les informations contenues par le contexte.

La première utilisation que nous étudierons est la prise de décision. Nous verrons comment se servir du contexte personnel d'un agent pour pouvoir décider de la meilleure action à réaliser.

La seconde application proposée est de fournir une assistance pédagogique à un apprenant. A partir de l'observation de ses actions, nous proposons de mettre en évidence les éléments importants qu'il n'a pas pris en compte dans sa prise de décision s'il commet une erreur. Nous verrons alors comment le contexte permet l'identification des éléments ayant échappés à l'apprenant, et permet la détection et l'explication de ses erreurs.

Basé sur notre étude de la PBD, notre modèle se base sur les actions réalisées par l'agent dans la simulation. Nous allons donc proposer, comme le permettent les systèmes de PBD, une phase d'apprentissage à partir des actions réalisées.

Dans ce chapitre nous verrons comment, à partir des actions réalisées par un avatar⁷ dans la simulation, nous proposons l'apprentissage de stratégies.

Nous terminerons ce rapport en présentant le travail effectué pour l'implémentation de notre modèle. Ce chapitre sera l'occasion pour nous de présenter une première utilisation de notre modèle, dans un environnement virtuel destiné à la formation au football.

Nous verrons les problèmes que nous avons rencontré, les solutions que nous proposons et pour finir les objectifs pour un travail ultérieur.

Avant d'en arriver à nos résultats, commençons par voir les travaux effectués dans le domaine de la programmation par démonstration.

⁷La représentation d'un humain dans un monde virtuel

Chapitre 2

Bibliographie

Les systèmes de PBD sont utilisés dans différents domaines comme la détection d'intrusion, la robotique ou les IHM *intelligentes* :

- Dans le domaine de la détection d'intrusion, le système de PBD va établir un plan représentant le comportement usuel des utilisateurs. Si plus tard, l'utilisateur exécute des actions qui sortent du plan, alors le système suppose que l'utilisateur a un comportement délictueux.
- Dans le domaine de la robotique, les systèmes de PBD vont permettre à un système mécanique, de reproduire des mouvements ou des réactions, par l'observation d'exemples.

Cette bibliographie, se limite à l'étude de la PBD dans les IHM *intelligentes*.

Dans un premier temps nous aborderons l'intégration et les buts de la PBD, vis à vis des IHM *intelligentes*.

Nous nous attarderons ensuite sur la création du modèle. Nous verrons les différents types de plans créés, et les comportements qu'ils permettent de modéliser. Nous attacherons ensuite une attention particulière à deux critères nous semblant important dans l'intégration de la PBD dans un EVF :

- Le système de PBD doit pouvoir créer les modèles comportementaux à partir d'un petit nombre d'itérations de l'exercice. En effet un apprenant ne voudra certainement pas reproduire des centaines de fois un exercice.
- La complexité des plans que l'on va pouvoir générer nous a aussi semblé primordiale. Est-ce que l'on va pouvoir abstraire une procédure, inclure des structures de contrôle? Dans le cadre d'un exercice, cela peut nous permettre de généraliser une même procédure dans deux contextes différents.

A ce stade, nous aurons cerné la création du plan représentant les modèles comportementaux. Nous étudierons alors les possibilités d'utilisation de ces plans. Pour cela, nous étudierons ce que peuvent restituer les systèmes de PBD et leurs capacités d'adaptation à l'utilisateur :

- Le contenu restitué par le système de PBD est important. Est-ce que l'on pourra prévoir une ou n actions, connaître les préférences de l'utilisateur, permettre la ré-exécution d'une procédure?
- Si l'on veut pouvoir appliquer des modèles comportementaux généraux, parce que chaque apprenant est différent, il faut que notre système soit capable d'adapter son plan à la personne.

Commençons dès maintenant par voir l'intégration et les buts de la PBD dans les IHM *intelligentes*.

2.1 Programmation Par Démonstration et IHM

L'informatique ayant un public de plus en plus large, les demandes en fonctionnalités sont de plus en plus nombreuses. Développer un logiciel pour chaque besoin serait impossible. Pour répondre à ce problème les logiciels intègrent de plus en plus de fonctionnalités. Se pose alors le problème de la personnalisation des applications.

Une première méthode, la plus simple, est celle des préférences configurables. L'utilisateur ira les modifier à sa guise. Cependant l'étendue des préférences est limitée à ce que le concepteur de l'application a jugé utile.

Plus proche de la PBD, nous allons retrouver les macros. L'utilisateur enregistre une séquence d'actions, qu'il pourra ensuite demander au programme d'exécuter. Le problème est que lors de petites variations dans la séquence le programme est inutilisable.

D'un niveau supérieur, les langages de scripts vont permettre aux utilisateurs expérimentés un contrôle maximal sur l'application. Dans ce cas, une expérience en programmation, une connaissance du langage et de l'interface avec le logiciel sont requises.

Le but de la PBD est d'allier la simplicité de création des macros avec les possibilités des langages de script. Ainsi, comme les macros, la PBD permet à l'utilisateur de construire son programme à partir d'une ou plusieurs séquences d'actions qu'il va réaliser. L'amélioration apportée est la possibilité pour la séquence de contenir des variations.

2.1.1 Généralités

De nombreux systèmes de PBD ont été développés depuis PYGMALION en 1975. En témoignent l'ouvrage de Allen Cypher [9], qui déjà référence 18 systèmes de PBD en 1993 et l'article [12] qui cite une vingtaine d'autres systèmes de PBD.

Si les systèmes de PBD varient du tout au tout, ils répondent toujours à trois critères :

- **La création d'un programme** : Le système de PBD va en effet extraire un plan (ou programme) des actions de l'utilisateur dans l'environnement étudié (en général limité à une application).
- **L'invocation d'un programme** : Une fois créé, le programme doit être invoqué : soit manuellement par l'utilisateur, soit automatiquement. Dans ce dernier cas le système doit déterminer lui-même les facteurs déclenchant.
- **L'exécution du programme** : Une fois que l'on sait quand il faut l'exécuter, reste à appliquer le plan au contexte actuel.

L'intégration des systèmes de PBD dans leur environnement logiciel est représentée figure 2.1. Les algorithmes ont toujours accès aux *actions* réalisées par l'utilisateur. Le *contexte* peut être une information supplémentaire. En général, la flèche *annotations* n'est pas présente, mais afin d'améliorer leurs performances, les systèmes peuvent demander un retour de la part de l'utilisateur. La flèche *exécution*, optionnelle elle aussi, permet au système l'exécution d'actions par lui-même.

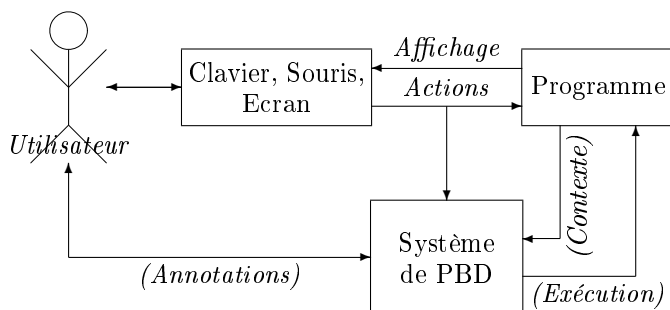


FIG. 2.1 – Intégration des systèmes de PBD

Selon le domaine d'application, les nécessités de l'application, les avancées des recherches, les systèmes de PBD peuvent présenter différentes caractéristiques plus ou moins développées lors de la création des plans et de leur utilisation.

2.1.2 La création du plan

La phase d'apprentissage peut prendre plusieurs formes : avec ou sans retour de l'utilisateur, type du plan généré, domaine étudié. Malgré ces différences, plusieurs critères se détachent lors de l'étude des systèmes de PBD :

- Le type du plan généré est important. En effet il va déterminer les capacités de modélisation. Nous positionnerons donc les différents types de plans les uns par rapport aux autres.
- Parce qu'on est face à un utilisateur, la rapidité de l'apprentissage est important : de une itération pour ACTIONSTREAMS[16] à une soixantaine pour COMETS[11]. Nous verrons donc les critères qui influent sur la rapidité de l'apprentissage.
- Vient ensuite la faculté de généralisation. Si une même procédure peut s'appliquer à l'identique sur deux instances différentes (par exemple copier/coller deux fichiers différents) le système de PBD peut réagir de deux manières : créer un plan pour chaque instance (ici les fichiers) ou créer un plan qui prendra en paramètre l'instance visée. Après avoir vu les procédés permettant l'abstraction, nous continuerons sur les techniques permettant de modéliser des structures de contrôle (branchements conditionnels, boucles, ...).

2.1.3 L'utilisation du plan

Une fois le plan créé, et quelle que soit la forme qu'il prenne, il faut maintenant l'utiliser. Nous verrons dans un premier temps à quoi peuvent être utilisés les plans précédemment créés puis les différents critères qui vont permettre de choisir un algorithme :

- L'utilisation que fait le système du plan généré varie de la simple prédiction de l'action suivante[11] à l'exécution de procédures[15] en passant par l'assistance[13]. Nous nous intéresserons aux limites et aux capacités induites par le type du plan.
- La phase d'apprentissage étant voulue assez courte, il est peu probable que le système de PBD ait été confronté à tous les cas possibles. Il peut donc arriver, lors de l'utilisation du plan, que le système de PBD se trouve confronté à une nouvelle situation ou à un dilemme. Il peut alors se bloquer[11]. Il peut aussi, par sa construction[13, 18], ou par la prise en compte d'éléments extérieurs[1] (préférences, statistiques, ...) déterminer ce que l'utilisateur va avoir tendance à choisir.

Même si les systèmes de PBD appliqués aux IHM *intelligentes* suivent tous une même procédure : création de plan puis utilisation du plan, nous allons voir dans les parties suivantes ce qui les distinguent, en commençant par le coeur du système : le plan modélisant les comportements de l'utilisateur.

2.2 Nature des plans générés

Comme nous l'avons vu dans la présentation il existe de nombreux systèmes de PBD. Bien que différents, les plans générés lors de l'apprentissage peuvent se diviser en deux grandes familles :

- **Les plans déterministes** : à la fois les plus anciens et les plus répandus. Ils prennent des formes variées, des plus simples (séquences[11]) aux plus complexes (arbres[16], règles logiques[14]).
- **Les plans non-déterministes** : ces plans sont apparus ces dernières années[13].

Nous étudierons donc dans cette partie les différents types de plans rencontrés.

Dans la description des plans déterministes, nous irons du plan le plus simple au plan le plus complexe.

Pour la partie sur les plans non-déterministes nous n'étudierons que les IOHMMs, une extension des chaînes de Markov. Cependant, l'utilisation des IOHMMs dans les systèmes de PBD n'est pas encore complètement décrite et certaines propriétés ne sont pas encore expliquées.

L'article [18] n'a pas été gardé car l'application de l'algorithme s'éloignait trop de la PBD et des IHM *intelligentes*. Cependant le modèle proposé, sous forme de diagrammes d'influences — une extension des réseaux bayésiens —, pourrait s'avérer intéressante.

2.2.1 Plans déterministes

SÉQUENCES D' ACTIONS Les informations qui vont nous intéresser dans la PBD sont le contexte dans lequel se trouve l'utilisateur (l'état du programme dans le cas de la PBD) et les actions qu'il va réaliser dans ce contexte.

L'algorithme COMETS[11] met en oeuvre cette structure avec la plus grande simplicité. L'environnement étudié est le jeu Space Invader. L'algorithme récupère l'état dans lequel se trouve le joueur (caché, en danger, ...) et l'action qu'il réalise (attaquer, se cacher, se décaler, ...). Tout au long de la partie, COMETS va stocker la succession d' {état, action} se présentant. A la fin de la partie il va découper ce log en sous-plans de quatre couples état-action. Tous les sous-plans apparaissant plus de cinq fois seront retenus. A partir de tous les sous-plans retenus, COMETS obtient une bibliothèque de sous-plans (cf. figure 2.2). Le problème principal de COMETS est qu'il génère des sous-plans de taille fixe.

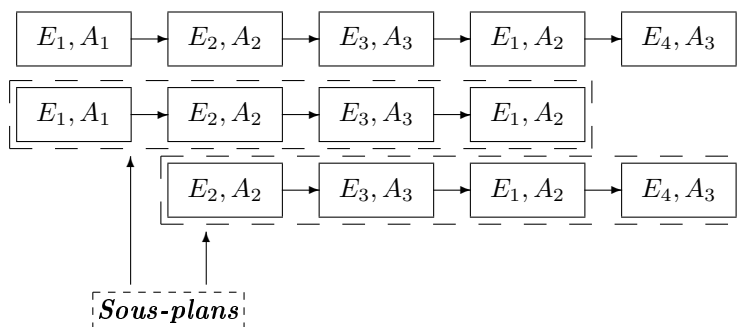


FIG. 2.2 – Création de la bibliothèque de sous-plans dans COMETS

Contrairement à COMETS qui génère des sous-plans de longueurs fixe, ACTIONSTREAMS va générer une bibliothèque de sous-plans de longueurs variables.

ACTIONSTREAMS[16] prend en comptes les actions de l'utilisateur sur toutes les applications et sur tout le système. Cependant, il ne prend pas en compte l'état dans lequel se trouve l'utilisateur.

La structure d'ACTIONSTREAMS est récursive, il pourra ainsi modéliser le fait qu'une tâche peut contenir des sous-tâches et des actions.

A partir du log d'une journée d'utilisation, il va découper la séquence d'actions en sous-plans de la manière suivante : soient $A_1 \dots A_n$ des actions et $S_1 \dots S_n$ des séquences d'actions, si S_1 est l'ensemble des actions de la journée alors :

$$S_1 = \{A_1, S_2, A_5, S_2, \dots\}, S_2 = \{A_2, A_3, A_4, S_3\}$$

Nous pouvons remarquer que S_2 est contenu plusieurs fois dans S_1 . S_2 contient lui-même des actions : A_2, A_3, A_4 et une sous-tâche : S_3 .

Malgré sa structure complexe, ACTIONSTREAMS est tout de même limité car il est incapable de généraliser une procédure appliquée à des éléments différents : Par exemple déplacer par copier-coller deux fichiers différents.

RÈGLES LOGIQUES DU PREMIER ORDRE Un des deux algorithmes comparés dans l'article de T. LAU[15], FOIL, est utilisé pour déterminer des règles PROLOG. Ces règles modélisent la succession des actions de l'utilisateur.

Le fonctionnement de FOIL[17] est basé sur la généralisation de règles. Son but est donc de trouver les règles qui font qu'un prédicat est vrai. Ces règles sont toutes vérifiées quand le prédicat est vrai et réciproquement au moins une règle n'est pas vérifiée quand le prédicat est faux.

On peut trouver une implémentation de FOIL sur internet avec plusieurs exemples¹.

FOIL répète les étapes suivantes :

- Initialiser deux ensembles vides
- Séparer les exemples en deux ensembles : les positifs vérifiant la règle (*vrais positifs*) et les négatifs vérifiant la règle (*faux positifs*)
- Créer une nouvelle règle qui permettra d'éliminer le plus de *faux positifs*
- Recommencer jusqu'à ce qu'il n'y ait plus de *faux positifs*
- Renvoyer l'ensemble des règles trouvées

Le système de PBD basé sur FOIL, détermine le prédicat *command*, permettant de déterminer l'action suivante, en fonction de règles basées sur l'action précédente.

Par exemple, à partir de la séquence d'actions suivante, FOIL va pouvoir déterminer les prédicats *command*.

Séquence d'actions :

1. Sélectionner A
2. Copier A
3. Coller A
4. Sélectionner un élément quelconque ($_$)

Prédicats générés par FOIL :

```
// On choisit un élément quelconque après avoir copié un élément quelconque
command(_, select) :-
    prevcommand(_, paste).

// On colle A après l'avoir copié
command(A, paste) :-
    prevcommand(A, copy).

// On copie A après l'avoir sélectionné
command(A, copy) :-
    prevcommand(A, select).
```

Sachant maintenant que l'action précédente était la sélection du fichier *test.txt* (`prevcommand('text.txt', select)`) on sait que la commande suivante sera de copier le fichier (`command('text.txt', copy)`).

La génération de règles logiques par FOIL est assez rapide. Cependant, une séquence contenant du bruit peut très vite détériorer le plan.

2.2.2 Plans non-déterministes

INPUT-OUTPUT HIDDEN MARKOV MODELS Ce type de plan, utilisé dans Sheepdog[13] et décrit dans l'article [5], permet la modélisation d'une séquence d'actions[4] de manière probabiliste.

Les HMMs (FIG. 2.3) sont constitués d'états (E_1, E_2, E_3). Ces états sont reliés entre eux par des arcs pondérés par une probabilité de passage

$$P(\text{Successeur}(E_i) = E_j) \in \{0, 1\}$$

¹Sources disponibles sur ftp://ftp.gmd.de/gmd/mlt/ML-Program-Library/foil/

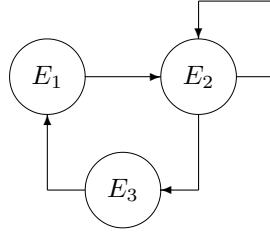


FIG. 2.3 – Représentation graphique d'un HMM

S'il n'y a pas d'arc la probabilité de passage est égale à 0.

Si à un instant t , le système se trouve dans un état E_i , alors à l'instant $t+1$ et en suivant les lois de probabilités fournies sur les arcs, le système se trouvera dans un nouvel état E_j avec j différent ou égal à i .

A chaque instant $(t, t+1, \dots)$, le HMM va donner une valeur à la sortie. Cette valeur est fonction de l'état dans lequel se trouve le HMM.

$$output_t = f(E_i)$$

Elle peut prendre une valeur symbolique (ici A, B ou C) par exemple et suivre une loi probabiliste :

	États		
	E_1	E_2	E_3
$P(A)$	0.2	0.0	0.5
$P(B)$	0.5	0.2	0.1
$P(C)$	0.3	0.8	0.4

En considérant que E_1 est l'état initial, on peut obtenir en sortie la séquence de lettres suivantes :

$$\{B, C, C, A, B, A, A, C, \dots\}$$

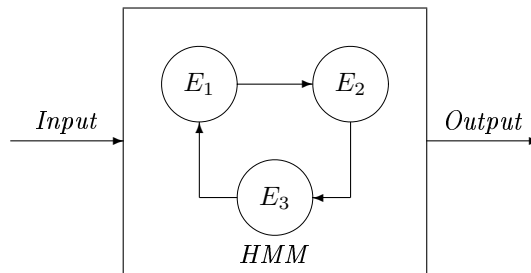


FIG. 2.4 – Représentation graphique d'un IOHMM

Les IOHMMs (FIG. 2.4) étant une extension des HMMs, on va retrouver les états reliés entre eux par des arcs pondérés et la production d'une sortie.

Au niveau structurel on rajoute l'entrée (*input*) : $u_t \in \mathfrak{R}^m$ et on change la sortie (*output*) en $y_t \in \mathfrak{R}^r$, m et r étant des entiers quelconques. La structure contenant les états et les transitions reste. Le nombre d'état est fixé à un nombre n lors de la création de l'IOHMM.

L'article ne précise pas comment est modélisé le contexte de l'IHM pour s'appliquer aux entrées réelles de l'IOHMM, ni comment l'action proposée est modélisée pour correspondre à des valeurs réelles en sortie.

Au niveau fonctionnement, on va retrouver plusieurs changements permettant l'utilisation de la valeur d'entrée (cf. Fig. 2.5) :

- Les probabilités de passage d'un état à un autre sont fonction de la valeur d'entrée :

$$N_i = P(\text{Successeur}(E_i) = E_j) = f(\mathbf{input}), f_{in}(\mathbf{input}) \in \{0, 1\}$$

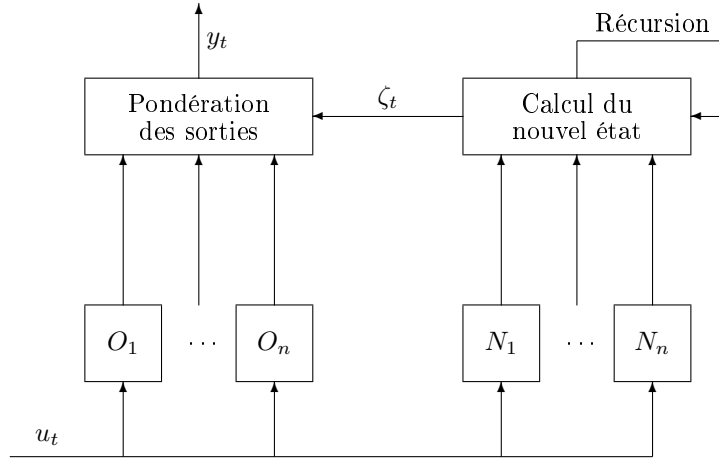


FIG. 2.5 – Structure interne d'un IOHMM

- La valeur de la sortie des états n'est plus seulement exprimée en fonction de l'état mais aussi de la valeur de l'entrée :

$$O_i = output_{E_i} = f_{out}(\mathbf{input}, E_i)$$

- L'état de l'IOHMM n'est plus déterminé de manière précise mais définit par un état global ζ_t , représentant l'appartenance à chaque état. Pour déterminer le nouvel état global ζ_t , l'IOHMM se sert de ζ_{t-1} et des N_i .
- Pour déterminer sa valeur de sortie, l'IOHMM pondère les différentes valeurs $output_{E_i}$ calculées précédemment par l'état global ζ_t .

On obtient donc un système complètement paramétré par la valeur d'entrée avec :

$$\zeta_t = f(\zeta_{t-1}, input)$$

$$output = f(\zeta_t, input)$$

On peut remarquer que ζ_{t-1} intervient dans le calcul de ζ_t . L'IOHMM va ainsi garder une *trace* ou influence des anciens états.

L'apprentissage sera chargé de modifier les fonctions f_{in} et f_{out} , ce qui modifiera la réaction de l'IOHMM face aux entrées successives. La nature des fonctions f_{in} et f_{out} employée n'est pas précisée.

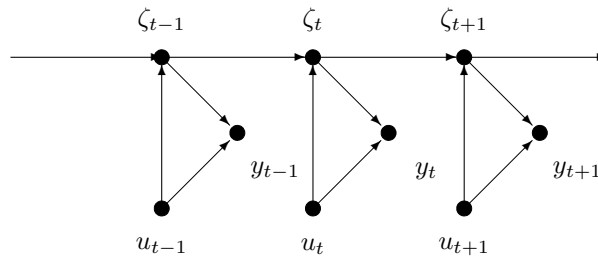


FIG. 2.6 – Influences entre les différents paramètres d'un IOHMM

La figure 2.6 illustre le fonctionnement dans le temps d'un IOHMM. L'état global courant est influencé par l'état global précédent et la valeur en entrée. La valeur en sortie est déterminée par l'état global courant et la valeur en entrée.

L'utilisation des IOHMMs étant à ses débuts dans le domaine des IHM *intelligentes*, il est dommage de ne pas trouver plus de détails sur leur utilisation.

Nous venons de voir les différents types de plans pouvant être générés par les systèmes de PBD. Ces plans, modélisant les comportements de l'utilisateur, peuvent aller de la simple séquence d'action à une structure probabiliste complexe tels les IOHMMs en passant par la création de règles logiques.

Nous allons maintenant voir les performances en terme de rapidité d'apprentissage des différents algorithmes.

2.3 Rapidité d'apprentissage

La première étape dans un système de PBD va être la création du plan. Il va ainsi créer, modifier, ajouter ou supprimer des plans afin de faire correspondre le plan final au comportement de l'utilisateur.

Afin de pouvoir aider l'apprenant, il faut que le modèle soit créé le plus rapidement possible, afin d'assister l'utilisateur dès le début de son apprentissage.

Si l'on veut un système de PBD opérationnel au plus vite, il faut alors que cette phase soit la plus courte possible. Nous allons donc voir sur différents systèmes de PBD les critères qui vont influencer la rapidité de l'apprentissage.

2.3.1 Les résultats obtenus

Pour COMETS, la phase d'apprentissage est définie arbitrairement à trois parties du jeu Space Invader[11]. Une fois ces trois parties terminées, le plan créé n'est pas mis à jour lors des parties suivantes.

L'apprentissage d'ACTIONSTREAMS nécessite toute une journée d'observations[16]. L'article ne précise pas si des modifications ultérieures peuvent être apportées ou pas au plan généré. ACTIONSTREAMS ne demande qu'une seule exécution, cependant sa durée s'étalant sur un jour, on ne peut pas vraiment considérer qu'il soit le plus rapide.

Le principal problème pour la rapidité d'ACTIONSTREAMS ce n'est pas sa modélisation mais le fait que son environnement soit le système entier. En effet plutôt que de se concentrer sur l'utilisation d'une application, il va falloir attendre la répétition des interactions de l'utilisateur avec une même application. Ici le problème vient donc du fait que l'on doit attendre que l'utilisateur revienne dans l'application. Il faut donc une durée suffisamment longue pour que l'utilisateur ait le temps, par exemple, de faire plusieurs copies de fichiers.

Un système de création de règles logiques, comme FOIL[15], n'a pas besoin d'une longue durée d'apprentissage. A partir d'un exemple, il peut créer une règle. Cependant cette règle aura de fortes chances de se révéler fautive par la suite.

Pour que la règle soit la plus précise possible, il faut multiplier les exemples. Ainsi le système de PBD va affiner les paramètres de ses règles et obtenir un meilleur plan. L'article ne précise pas si les règles sont modifiées une fois la phase d'apprentissage terminée. Pour un algorithme comme FOIL, la durée de la phase d'apprentissage n'a d'importance que compte tenu de la précision que l'on veut obtenir.

Le système de PBD SHEEPDOG, basé sur les IOHMMs, a été entraîné par onze experts qui avaient un même but : la configuration d'une machine sur un réseau. Chaque expert disposait d'une configuration différente[13]. Aucune modification n'est apportée au plan une fois l'apprentissage terminé. Cependant, il est précisé que, dans le futur, SHEEPDOG pourra avoir un retour sur ses actions afin de pouvoir améliorer son plan sans l'aide d'un expert.

Dans le cas de SHEEPDOG, la multiplication des exemples n'est pas nécessaire. Il peut déterminer un plan à partir d'un seul exemple. Cependant, multiplier les exemples (dans différentes configurations) va lui permettre également d'affiner son plan pour pouvoir mieux répondre à un plus grand nombre de situations.

2.3.2 Les critères

D'après les observations précédentes, on se rend compte que l'algorithme influence peu la durée de l'apprentissage, car leur but est d'être rapide.

Au contraire, c'est l'environnement dans lequel se situe le système de PBD, la précision que l'on veut en sortie et les critères d'évolution du plan qui vont déterminer le nombre d'exemples nécessaires pour obtenir le résultat attendu.

Si tous les algorithmes, dans leurs environnements, peuvent apprendre rapidement, nous allons voir maintenant que leur capacité à généraliser les procédures de l'utilisateur sont très variables.

2.4 Généralisation

Lors de la phase d'apprentissage, le but est de confronter le système de PBD à un maximum de situations. Il pourra ainsi, s'il est capable d'abstraire suffisamment les procédures, créer un plan lui permettant de réagir à un maximum de cas.

Nous allons voir maintenant les techniques utilisées pour, à partir de plusieurs exemples différents, obtenir un plan capable de reproduire les actions de l'utilisateur.

2.4.1 Généralisation de paramètres

ACTIONSTREAMS n'est pas capable de généraliser une même procédure appliquée à deux fichiers différents (ouvrir, éditer, fermer par exemple). C'est un réel problème car on multiplie le nombre de procédures par le nombre de fichiers sur lesquels elle est appliquée. L'auteur propose alors l'utilisation de contextes. Il faudra alors généraliser le nom de fichier (par exemple : *.txt) pour la procédure. Cependant ce mécanisme n'est pas encore implémenté.

Le problème de la généralisation de paramètres est aussi abordé par SHEEPDOG. Les auteurs proposent de prendre le problème d'un autre point de vue.

Les systèmes de PBD déterministes considèrent que la procédure est fixe (la séquence d'actions est toujours la même) et que les paramètres des actions sont variables (fichier différent par exemple). Les auteurs de l'article, proposent de considérer que les paramètres des actions sont relativement simples (on regroupe donc l'action et ses paramètres), mais que la procédure varie considérablement, à cause des branchements conditionnels[13].

La généralisation ne se fait donc plus sur les paramètres mais sur les actions.

De par sa structure, FOIL est le seul à réellement prendre en compte la généralisation de paramètre dans son plan. En effet, la création des règles fournit le mécanisme idéal pour la généralisation puisqu'elles prennent des paramètres en argument.

FOIL va, en plus, proposer une option, le *Closed World Assumption*, qui va permettre de limiter la généralisation des paramètres aux exemples fournis. Si le CWA est activé, les règles ne doivent pas permettre de générer d'actions qui n'ont pas été données dans les exemples.

2.4.2 Structures de contrôles

ACTIONSTREAMS organise son plan en tâches et sous-tâches, chaque sous-tâche étant une séquence d'actions. Afin de complexifier son modèle, l'auteur de l'article sur ACTIONSTREAMS propose des extensions au modèle[16].

La première est la possibilité d'avoir des séquences d'actions non ordonnées. En effet, l'utilisateur peut réaliser des actions dans un ordre indépendant du bon déroulement de la tâche.

Ensuite, il propose de pouvoir rendre certaines tâches conditionnelles ou facultatives. Il propose une structure permettant l'introduction de points de choix lors de l'exécution de la procédure. Cependant deux problèmes se posent : où introduire un point de choix et comment décider des conditions.

La dernière extension proposée par l'auteur est la reconnaissance de boucles. Déterminer combien de fois l'action a été répétée est facile, mais déterminer la condition d'exécution ou d'arrêt de la boucle reste un problème complexe.

FOIL propose une réelle solution aux branchements conditionnels. En effet lors de la création des règles, il peut en obtenir qui, dans un langage procédural, représentent un branchement conditionnel. Par exemple :

```
command(A, delete)
    :- prevcommand(A, select), sender(A, pub).
command(A, archive)
    :- prevcommand(A, select), sender(A, banquier).
```

correspond à :

```
A.select();
if (A.sender == pub) A.delete();
if (A.sender == banquier) A.archive();
```

Comme nous l'avons vu précédemment, SHEEPDOG ne propose pas de généralisation de paramètres. L'approche proposée est de générer un plan incluant des branchements conditionnels et des boucles[13].

Dans SHEEPDOG, les structures de contrôle vont être représentées par les changements d'état fonction de l'entrée. L'article ne précise pas comment est représenté le système en entrée de l'IOHMM mais on sait qu'il se base sur l'état courant (la fenêtre au premier plan) pour déterminer ce qu'il doit faire. Un changement d'état pondéré différemment en fonction des entrées va représenter un branchement conditionnel. Une boucle peut être représentée par un retour au même état en suivant un ou plusieurs branchements conditionnels.

Nous avons vu que les systèmes de PBD pouvaient apprendre vite et généraliser les procédures exécutées par l'utilisateur. Nous allons maintenant nous pencher sur l'utilisation des plans précédemment créés.

2.5 Restitution

Une fois le plan créé, qu'il soit déterministe ou non-déterministe, simple ou complexe, les systèmes de PBD doivent l'utiliser.

Si les environnements ciblés par les systèmes de PBD peuvent aller de la simple application à tout le système, leurs utilisations des plans générés varient aussi. Nous allons donc voir les différents objectifs que se fixent les systèmes de PBD.

2.5.1 Adaptation à l'utilisateur

Le plan modélise le comportement de l'utilisateur. Il va permettre aux systèmes de PBD de pouvoir fournir à tout moment, une information sur les intentions de l'utilisateur.

COMETS se sert de son plan pour déterminer la prochaine action du joueur en fonction de ses trois dernières actions et de son état courant. L'intégration d'un système de PBD dans un jeu doit permettre le développement d'entités autonomes plus réalistes. La faculté d'anticipation des actions de l'utilisateur va permettre aux personnages d'adapter leurs comportements[11]. On évite ainsi un jeu statique où tout se reproduit à l'identique à chaque partie.

Dans le domaine des IHM, ACTIONSTREAMS utilise son plan pour déterminer les actions que l'utilisateur a l'habitude de réaliser. L'auteur propose deux exemples d'utilisations.

Le premier est la prédiction d'action. Grâce au plan et à l'action courante, il est possible d'estimer ce que va faire l'utilisateur par la suite. Sans servir à exécuter une procédure, cette prédiction peut être utile pour pré-charger des boîtes de dialogues ou autres fenêtres en fond de tâche[16].

Le second exemple est la personnalisation de l'interface utilisateur. A partir du plan il sera possible de déterminer les actions préférées de l'utilisateur. On peut aussi extraire du plan les préférences pour l'accès aux commandes de l'utilisateur (menu, icône). L'intérêt est de pouvoir mettre à disposition de l'utilisateur tout ce dont il aura besoin fréquemment et sous la forme qui lui conviendra le mieux[16].

2.5.2 Assistance

Si la phase d'apprentissage d'un PBD est basée sur l'observation d'un expert, alors le plan généré va pouvoir être utilisé en premier recours comme assistance à l'utilisateur dans le domaine de l'expert. Quand l'utilisateur sera confronté à une situation qu'il ne connaît pas, il pourra faire appel au système de PBD pour lui proposer une procédure à réaliser pour atteindre son but.

Dans l'optique d'apporter une aide pour le support technique, SHEEPDOG va se placer en expert dans le système et aider l'utilisateur à configurer ses paramètres réseau.

L'équipe de SHEEPDOG note que les scripts ou la documentation sont souvent insuffisants ou obsolètes face aux besoins de l'utilisateur. La plupart des procédures de maintenance ont besoin d'une intervention humaine à un certain moment. Ce point n'est d'ailleurs pas toujours atteignable automatiquement, en cas d'erreur par exemple.

SHEEPDOG n'a pas été conçu pour réaliser la tâche automatiquement mais pour assister l'utilisateur. Dans une fenêtre il va donc lui proposer une action à réaliser. Après avoir évalué le résultat de l'exécution de l'action, SHEEPDOG propose une autre action et ainsi de suite.

2.5.3 Automatisation de tâches

Dans certains cas, le système de PBD peut être mis en place de manière à remplacer un système de macros. Dans ce cas il va exécuter lui même la procédure.

Le système développé par A. CYPHER, EAGER, est chargé d'automatiser les tâches répétées de l'utilisateur. Ce système de PBD va permettre d'automatiser des tâches dans HyperCard². EAGER va examiner continuellement les actions réalisées par l'utilisateur.

²Un système de base de données intégré dans MAC OS, voir <http://en.wikipedia.org/wiki/HyperCard> pour plus de détails

Lorsqu'il détecte la répétition d'une tâche, il va proposer à l'utilisateur de l'exécuter automatiquement. Pour cela, il va d'abord faire valider la séquence d'actions qu'il veut réaliser par l'utilisateur puis la répétera automatiquement s'il y a lieu[8].

Comme nous venons de le voir, les systèmes de PBD vont permettre de déterminer les intentions de l'utilisateur. Elles se serviront ensuite de cette prévision pour adapter l'environnement à l'utilisateur, assister l'utilisateur ou automatiser les tâches de l'utilisateur. Nous allons maintenant voir comment les systèmes de PBD vont pouvoir s'adapter à de nouveaux utilisateurs ou à de nouveaux comportements.

2.6 Adaptation du plan

Comme nous l'avons vu précédemment, les systèmes de PBD vont fournir plusieurs types de services. Si grâce à la structure de leur plans les systèmes non-déterministes proposent toujours une action[13], il n'en est pas de même pour les algorithmes basés sur des plans déterministes[1, 11, 16].

Nous allons donc effectuer un tour d'horizon des différentes méthodes utilisées pour adapter le plan à l'inconnu. Il s'agit, pour le système de PBD, de trouver une proposition dans toutes les situations, même si elle n'a pas été rencontrée auparavant.

2.6.1 Modifications manuelles

Lors de la création du plan, certains systèmes permettent des annotations de la part de l'utilisateur. Le but est de permettre à l'utilisateur d'affiner manuellement le plan proposé par le système de PBD.

Même si le type de plan fourni par ACTIONSTREAMS n'en permet pas la mise en oeuvre, D. MAULSBY pose le problème de l'ordonnancement des tâches, des tâches conditionnelles et du choix entre plusieurs tâches[16].

La solution proposée par ACAPPELLA est de permettre à l'utilisateur de sélectionner les stimuli retenus par le système de PBD[10]. On obtient ainsi les préférences de l'utilisateur à propos des critères de déclenchement de la procédure.

SHEEPDOG, bien qu'autonome, propose lui aussi une interface permettant à l'utilisateur de modifier le plan généré lors de l'apprentissage. Grâce à une représentation sous forme graphique, l'utilisateur pourra déplacer les actions afin de modifier leur ordre d'exécution.

2.6.2 Statistiques et préférences

Un système de PBD comme COMETS ne fournit pas toujours de réponse. En effet lorsque plusieurs prévisions se présentent à lui, il est incapable de déterminer quelle est la plus appropriée. L'utilisation de statistiques et de préférences de l'utilisateur vont permettre de remédier à ce problème.

M. BAUER, propose une approche intéressante[1]. Sa méthode consiste à pondérer les plans valides en fonction du nombre de fois où ils se sont révélés valides.

Lors du premier conflit entre sous-plans, le système ne va pas pouvoir fournir de prévision. Cependant, lorsque le conflit se reproduit, il va estimer qu'il y a plus de chances que le plan valide la première fois soit de nouveau valide. Il va donc proposer — avec un certain degré d'incertitude — une prévision. Cette technique peut être assimilée à un renforcement des sous-plans les plus utilisés.

Cependant cette méthode seule ne permet pas de répondre s'il n'a aucune propositions. La solution proposée dans l'article est de comparer le cas présent aux autres cas et de prendre le plus proche.

On obtient ainsi une prévision pour chaque cas, qu'il ait été rencontré ou pas et qu'il ait plusieurs solutions ou pas.

Cependant un problème reste en suspend. Si l'utilisateur utilise dans la majorité des cas un plan A mais que dans un cas bien précis et plus rare il utilise un plan B, l'heuristique précédente va se précipiter sur le plan A à tort.

Afin de corriger ce problème, M. BAUER propose d'utiliser une technique s'apparentant à des préférences de l'utilisateur (ou dépendance du contexte). L'auteur se sert donc de l'état du système (par exemple l'émetteur et la taille d'un mail) pour différencier les contextes dans lesquels sont applicables les plans.

La phase d'apprentissage de l'algorithme va permettre la génération d'un arbre ID3 des paramètres qui permettra la pondération des plans en fonction des préférences de l'utilisateur.

On obtient ainsi un résultat à la fois dépendant des préférences de l'utilisateur et des statistiques sur le nombre d'apparitions du plan.

2.6.3 Changement de tâche

L'algorithme d'*Intention Guessing*[19] (IG) a pour plan une liste de *stratégies* (séquences d'actions) que l'utilisateur d'un lecteur de news peut réaliser. Le problème relevé, lors de l'étude d'un utilisateur, est qu'il peut changer de stratégie à tout moment : il peut passer de la lecture linéaire de news à la recherche d'un nouveau sujet, ou à l'approfondissement d'un autre sujet.

L'algorithme se base sur son plan pour calculer la probabilité qu'a un utilisateur de réaliser une stratégie, connaissant les dernières actions de l'utilisateur. Pour déterminer les probabilités, l'algorithme possède plusieurs critères[19] :

- Le nombre de stratégies correspondant aux dernières actions.
- Le nombre d'actions en accord avec chacune de ces stratégies.
- Une stratégie en cours a plus de chance de continuer qu'une nouvelle de commencer.
- Les stratégies revenant plus souvent (*cf. plus haut*) ont plus de chance d'être appliquées.

L'algorithme ne garde que les dernières actions (le nombre exact n'est pas précisé). Cela va permettre de laisser tomber une stratégie si elle ne convient plus.

L'auteur précise que l'algorithme présenté peut être combiné à un système de reconnaissance de plus haut niveau :

- la partie IG permettra d'abstraire les actions de l'utilisateur
- le second système reconnaîtra ce qu'est en train de faire l'utilisateur

Les mécanismes que nous venons de voir permettent donc de passer la dernière limite. Les systèmes de PBD peuvent maintenant utiliser des plans incomplets ou changer rapidement de plan, bref s'adapter aux changements de comportements de l'utilisateur.

Cette étude bibliographique a été l'occasion d'étudier les travaux menés dans le domaine de la PBD dans les IHM *intelligentes*. Nous allons maintenant voir le modèle que nous proposons et plus tard les utilisations que nous en avons.

Chapitre 3

Un modèle pour les comportements dynamiques

Nous avons pu constater à l'issue de l'étude bibliographique que l'état dans lequel se trouve l'utilisateur est un critère important pour l'identification et l'utilisation d'une procédure. Par exemple COMETS utilise un état global associé au vaisseau spatial, dans SHEEPDOG c'est un ensemble de paramètres correspondant à l'état du système qui va être utilisé.

Nous avons donc décidé que notre modèle de comportement devrait contenir des informations concernant l'état de l'apprenant. Pour cela nous nous sommes rattaché aux travaux de Bénard [3] proposant la modélisation de l'état du joueur par son contexte personnel.

Dans la section 3.1, nous reviendrons sur la notion de contexte. Nous aborderons également les moyens mis en oeuvre pour pouvoir utiliser ces contextes :

- La génération du contexte personnel de l'apprenant
- La comparaison de deux contextes

Finalement, dans la section 3.2, nous étudierons notre modèle : les graphes contextuels dynamiques. Pour effectuer notre modélisation, nous avons choisi de nous baser sur les graphes contextuels de Brézillon [7]. Nous verrons alors les modifications apportées à la syntaxe des graphes contextuels.

3.1 La notion de contexte

3.1.1 Définition

Brézillon définit, dans [7], la notion de contexte comme l'ensemble des conditions appropriées et autres influences qui font qu'une situation est unique et compréhensible. En nous appuyant sur cette définition, nous définissons le contexte comme un ensemble de perceptions. La définition d'une perception est donnée dans la section 3.1.2

- Nous détachons deux types de contextes :
- Le contexte personnel
 - Le contexte décisionnel

Le contexte personnel ne se rapporte pas à la définition donnée par Brézillon et correspond à l'état dans lequel se trouve l'apprenant. Le contexte personnel de l'apprenant va être constitué de l'ensemble de ses perceptions dans son environnement. Les perceptions contenues dans le contexte personnel ne sont pas filtrées.

Nous verrons dans la section 3.1.3 comment est généré le contexte personnel.

Le contexte décisionnel quand à lui ne contient que les perceptions appropriées à une situation. C'est ce type de contexte que nous utiliserons par la suite pour modéliser l'état nécessaire pour

réaliser une action.

Dans la section 3.1.4 nous verrons comment comparer le contexte personnel avec un contexte décisionnel. Le résultat de cette comparaison nous permettra de trouver le contexte décisionnel le plus approprié pour la situation courante.

3.1.2 La notion de perception

Définition

Bénard dans [2] propose une formalisation des perceptions que nous allons réutiliser.

Les perceptions peuvent être de différents types. Nous pourrions ainsi, dans un environnement virtuel, faire la différence entre une perception de distance et une perception de présence.

Une perception est définie comme la composition de 3 éléments propres à la perception et 1 élément propre au type de la perception :

- La *source* représente par exemple le point de départ d'une mesure.
- La *cible* représente le point d'arrivée de la mesure.
- La *valeur* permet de donner la mesure.
- La *plage de valeurs* permet de donner l'échelle de la mesure. Cette plage de valeur est propre au type de la perception, nous permettant ainsi de comparer les valeurs de deux perceptions du même type.

Afin de pouvoir effectuer une comparaison entre deux valeurs, on associe à chaque valeur un entier.

Comparaison

Pour pouvoir comparer deux perceptions, il faut que les perceptions soient du même type et aient la même source et la même cible.

Si deux perceptions peuvent être comparées alors on va déterminer leur différence suivant la formule suivante :

$$d(p1, p2) = \frac{|value_{p1} - value_{p2}|}{value_range_{type}.size() - 1}$$

Avec :

- $p1$ et $p2$ deux perceptions.
- $type$ le type de $p1$ et $p2$.
- $value_range_{type}$ le nombre de valeur que peut prendre une perception du type $type$.
- $value_{p1}$ et $value_{p2}$ leurs valeurs respectives de $p1$ et $p2$.

Ainsi si on compare une perception dont le value range est {false = 0, true = 1} on obtient :

- $d(pTrue, pFalse) = \frac{|1-0|}{1} = 1$
- $d(pTrue, pTrue) = \frac{|1-1|}{1} = 0$

Limitation

Nous allons maintenant voir un exemple de perception : la distance. Admettons que l'on définisse une plage de valeur comme suit :

- près : $< 5m$
- proche : $< 15m$
- loin : $< 200m$
- très loin : $\geq 200m$

Le fait que deux objets a et b soient séparés de 30m sera représenté par la perception :

$$distance(a, b, loin)$$

Un problème rencontré, lors de l'implémentation est que la perception est uni-directionnelle. En effet grâce à $distance(a, b, loin)$ le système sait que a est loin de b mais pas que b est loin de a . Pour

avoir le contexte le plus complet possible il faut alors avoir :

$$distance(a, b, loin), distance(b, a, loin)$$

Le nombre de perceptions que nous allons avoir dans le contexte personnel risque donc d'être conséquent. Pour limiter l'explosion du nombre de perceptions, il est recommandé d'appliquer un filtre sur les perceptions générées pour ne garder que celles réellement pertinentes.

3.1.3 Génération d'un contexte

Si l'apprentissage artificiel ou l'expérience d'un expert peut nous apporter la liste des perceptions pour un contexte décisionnel, nous allons voir dans cette section comment est généré le contexte personnel.

L'algorithme 1 décrit la génération du contexte personnel.

Algorithme 1 : Génération d'un contexte

```

1 Récupérer la liste des objets perçus;
2 Récupérer la liste des types de perceptions;
3 pour chaque Type de perception faire
4   | Générer la liste des perceptions à partir de la liste d'objets;
5 fin

```

Le première étape est de récupérer la liste des objets perçus. Cette liste doit contenir tous les objets sur lesquels on veut qu'une perception soit générée. On peut donc, par exemple, dégrader la prise de décision en ne tenant compte que des objets les plus proches.

Ensuite on va demander à chacun des types de perceptions de générer l'ensemble des perceptions de son type, ayant pour source et pour cible des objets de la liste précédemment générée.

En assemblant toutes les perceptions générées, on obtient le contexte personnel de l'apprenant.

3.1.4 Comparaison de deux contextes

La fonction de similarité entre deux contextes nous permet d'identifier les contextes décisionnels les plus proches du contexte personnel.

Cette mesure est une fonction qui est bornée entre 0 et 1. Une similarité égale à 1 indique que les deux cas sont similaires.

Cette fonction est de la forme :

$$Sim(C_s, C_c) = 1 - \sum_{i=1}^n w_i * |(p_i^s - p_i^c)|$$

Où C_s : le contexte courant, C_c : le contexte à comparer, p_i^s et p_i^c les perceptions de ces contextes et le poids de la perception pour ce contexte. $w_i * |(p_i^s - p_i^c)|$ est borné entre 0 et $1/n$, n étant le nombre de perceptions. Si une perception n'est pas présente, une pénalité égale à la différence maximale est infligée, soit $1/n$. Cette stratégie est appelée *pessimistic measure* dans [6].

3.2 Les graphes contextuels

Le problème principal avec le raisonnement à base de cas proposé dans [3] c'est que l'on ne dispose d'aucun moyen pour exprimer une succession temporelle d'actions pour représenter une stratégie. Pour le permettre, nous avons défini les Graphes Contextuels Dynamiques (Dynamic Contextual Graphs), basés sur les graphes contextuels de Brézillon [?].

Chaque Graphe Contextuel Dynamique (noté DCxG par la suite) représente une stratégie différente ou un moyen différent d'arriver à un même but. Un graphe contextuel est composé de noeuds instanciant des contextes, des actions ou des graphes. Les noeuds sont reliés entre eux par des arcs orientés.

3.2.1 Les différents types de noeuds

Les noeuds contextuels : Les noeuds contextuels sont des instances des contextes décisionnels. Ils seront représentés par des ellipses contenant le nom du contexte instancié (cf. FIG 3.1).



FIG. 3.1 – Un noeud instanciant le contexte décisionnel C0

Les noeuds d'actions : Une action atomique est une méthode exécutable, c'est à dire une action qui peut être réalisée en un temps fini.

Le noeud d'action va pouvoir prendre 3 formes :

1. Les noeuds d'actions instanciant des actions atomiques. Ils sont représentés par des rectangles contenant le nom de l'action instanciée (cf. FIG 3.2).



FIG. 3.2 – Un noeud instanciant l'action atomique A0

2. Les noeuds désignant une action cyclique sont représentés par un rectangle divisé en deux, la partie supérieure contenant l'action atomique et la partie inférieure la condition d'arrêt (cf. FIG 3.3).

Une action atomique peut ainsi être rendue cyclique. Elle est représentée entre parenthèses.

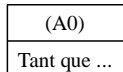


FIG. 3.3 – Un noeud rendant cyclique l'action atomique A0

3. Les noeuds appelant un autre graphe (appelé dans ce cas sous-graphe du graphe courant). Ces noeuds seront représentés par un trapèze contenant le nom du graphe appelé (cf FIG 3.4).



FIG. 3.4 – Un noeud appelant le graphe P2

3.2.2 Les liens entre les noeuds

Tous les liens entre les différents noeuds sont orientés, ils seront donc représentés par des flèches. Un lien entre deux noeuds est noté par le symbole \rightarrow .

La source du lien est le noeud d'où part la flèche et la destination, le noeud où va la flèche (cf. FIG 3.5).

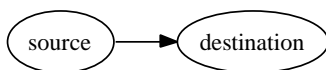


FIG. 3.5 – Un lien reliant deux noeuds

La source est un précédent de la destination et la destination est un suivant de la source. Par exemple dans $C0 \rightarrow A0$:

- C0 est la source et A0 est la destination
- C0 est un précédent de A0 et A0 est un suivant de C0

Nous allons maintenant voir les différents types de liens disponibles et leur sens dans le graphe.

Contexte → Contexte : Un lien allant d'un contexte vers un autre contexte représente une spécialisation du contexte source. C'est à dire que les noeuds suivant le second contexte auront pour contexte précédent la combinaison des deux contextes.

Contexte → Action : Un lien allant d'un contexte vers une action va définir le contexte précédent de l'action, c'est à dire, le contexte adéquat pour la réalisation de l'action.

Action → Action : Un lien d'une action vers une autre action définit un enchaînement d'actions. Le contexte précédent de la destination sera le même que celui de la source.

Action → Contexte : Un lien allant d'une action vers un contexte va permettre d'initialiser une nouvelle composition de contextes. Cela signifie que pour continuer la stratégie, le nouveau contexte sera important et on oublie l'ancien contexte.

Exemple : Pour cet exemple nous étudierons la séquence (cf. FIG. 3.6) :

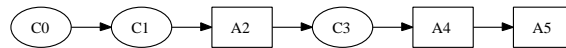


FIG. 3.6 – Un exemple simple

- C1 est une spécialisation de C0.
- A2 a pour contexte précédent la composition C0 + C1.
- Après A2, on commence une nouvelle composition de contextes.
- A4 a pour contexte précédent C3.
- A5 a pour contexte précédent C3.

3.2.3 Graphe contextuel

Un graphe contextuel est identifié par un nom qui doit être unique. Il est composé de noeuds, de liens et de trois contextes facultatifs. Ces trois contextes vont définir :

- Le contexte initial, nécessaire pour rentrer dans le graphe.
- Le contexte final, justifiant le bon déroulement du graphe.
- Le contexte d'exception, permettant une sortie immédiate du graphe.

Ils sont représentés dans un rectangle aux angles arrondis, découpé en trois parties (cf. FIG. 3.7). Dans la partie supérieure, nous retrouverons : à gauche le contexte initial, et à droite le contexte final. Dans la partie inférieure nous retrouverons le contexte d'exception.

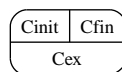


FIG. 3.7 – Les trois contextes liés au graphe

La figure 3.8 montre un exemple de graphe contextuel mettant en oeuvre les mécanismes présentés précédemment.

Le graphe s'appelle "Graph1". Il est composé de 8 noeuds : 4 contextuels, 4 actions et 7 liens.

Pour pouvoir rentrer dans ce plan il faut valider le contexte décisionnel C_{init} . Si on suit le graphe jusqu'au bout on doit se retrouver dans le contexte C_{fin} . Si jamais le contexte C_{ex} devient vrai quand on est dans le graphe, une exception sera relevée et on sortira immédiatement du graphe.

Maintenant que nous savons définir des graphes contextuels, il va nous falloir un système capable de les exploiter.

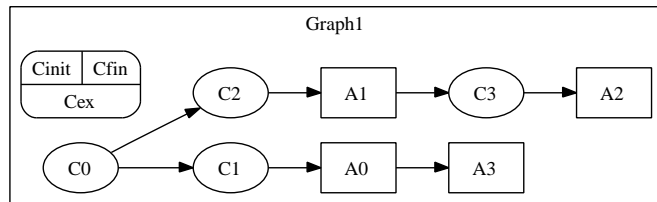


FIG. 3.8 – Un exemple de graphe

Chapitre 4

Utilisation du modèle

Nous venons de voir la définition des Graphes Contextuels Dynamiques (notés DCxG¹ par la suite). Maintenant que nous savons modéliser des comportements non-procéduraux, il nous faut des mécanismes capables d'exploiter les informations contenues.

Notre premier objectif va être de proposer un modèle d'algorithme, permettant l'exploitation des DCxG de manière générique. Il devra pour cela être capable de s'adapter à différentes utilisations.

Nous verrons ensuite comment spécialiser ces algorithmes pour permettre à un agent autonome d'agir dans la simulation. Les DCxG nous serviront dans ce cas de support d'information pour la prise de décision.

Dans la dernière partie nous verrons, comment, à partir des actions réalisées par un apprenant, nous pouvons utiliser les DCxG pour identifier la stratégie appliquée par l'apprenant. Les DCxG seront alors utilisés comme modèle pour une identification.

4.1 Mécanismes génériques

Nous allons commencer par étudier l'architecture que nous proposons. Nous commencerons par l'organisation des données, puis nous verrons les pointeurs et leurs gestionnaires. Les gestionnaires sont chargés de la liaison entre la simulation et les graphes. Nous évoquerons alors le couplage entre la simulation et notre mécanisme d'utilisation des graphes.

Nous étudierons ensuite les algorithmes proposés pour pouvoir avancer dans un graphe. La section suivante explique comment spécialiser ces algorithmes pour une prise de décision ou une aide pédagogique.

4.1.1 Organisation des graphes

Comme nous nous trouvons dans une situations dynamique, nous n'avons pas à notre disposition une procédure permettant de modéliser l'exercice du début à la fin. Nous aurons donc plusieurs DCxG à stocker pour modéliser le comportement global d'un apprenant. Il nous faudra donc organiser cette multitude de graphes.

Comme chaque apprenant ne jouera pas forcément le même rôle dans la simulation, nous avons décidé de diviser l'ensemble de nos graphes en bibliothèques de graphes. Notre espace de stockage sera donc celui présenté à la figure 4.1.

Chaque gestionnaire aura une bibliothèque associée. Elle lui permettra ainsi de puiser dans les stratégies correspondant au rôle qui a été attribué à l'agent.

¹Dynamic ConteXtual Graphs

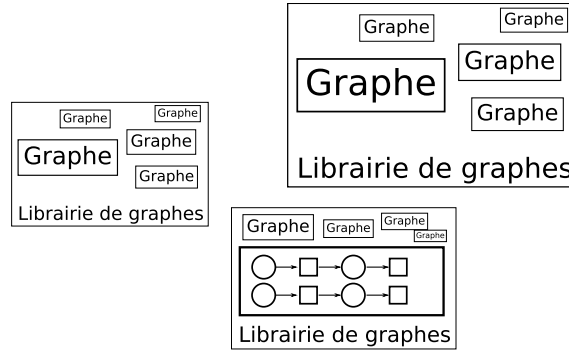


FIG. 4.1 – Structuration des données

4.1.2 Parcours et interprétation de graphe

Le mécanisme principal dans l'utilisation des graphes va être le pointeur. Comme leur nom l'indique, les pointeurs vont pointer sur un noeud ou un graphe (cf. FIG.4.2). Ils permettent de marquer une position dans un graphe.

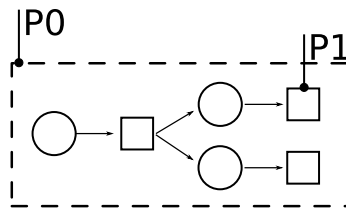


FIG. 4.2 – Pointeurs sur et dans un graphe.
P0 point sur le graphe tandis que P1 pointe sur une action.

Lors de sa création, le pointeur se verra attribué un graphe. A partir de ce moment son but sera de parcourir tous les chemins disponibles dans le graphe.

Afin de lui donner la possibilité de parcourir tous les chemins, le pointeur dispose d'une méthode lui permettant de se dupliquer. Ainsi il pourra envoyer d'autres pointeurs lors d'un embranchement pour pouvoir explorer tous les chemins sortants du noeud.

A chaque nouveau noeud rencontré, le pointeur pourra effectuer des modifications sur ses données (cf. section 4.1.3).

Afin de gérer la création et la destruction de pointeurs, mais aussi pour en extraire les informations qui nous intéressent, nous avons créé un gestionnaire de pointeur (cf. FIG. 4.3).

Liaison entre les graphes et la simulation, le gestionnaire de pointeurs va devoir gérer le cycle de vie de ses pointeurs. Il devra créer les pointeurs sur les différents graphes composant sa librairie et si nécessaire les détruire.

Ce sont le contexte personnel et les actions réalisées par l'agent (avatar ou autonome) qui vont permettre au gestionnaire de prendre les décisions sur le sort des pointeurs.

La position des pointeurs restant, donnera au gestionnaire les informations nécessaires pour qu'il puisse remplir sa fonction dans l'EVE. Afin de pouvoir retourner une information pertinente à l'EVE, le gestionnaire devra ensuite filtrer les informations qu'il aura récupéré des pointeurs.

Pour notre mécanisme d'utilisation des graphes, nous nous sommes imposé de pouvoir mettre en concurrence plusieurs DCxG. Cette contrainte implique que l'on pourra obtenir plusieurs propositions lorsque l'on va chercher la stratégie en cours ou l'action à réaliser.

Nous discuterons du choix de la meilleure stratégie dans les sections suivantes, concernant la spécialisation des gestionnaires.

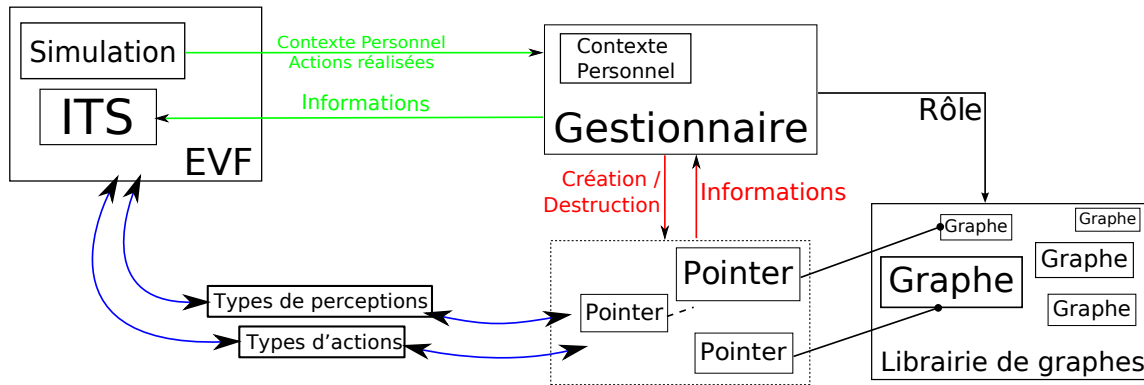


FIG. 4.3 – Gestionnaire de pointeurs

Afin de pouvoir retourner des informations à la simulation et pour garder une cohérence, les types de perceptions et les types d'actions réalisables doivent être partagés par la simulation et le gestionnaire.

Le fonctionnement d'un gestionnaire étant spécifique à l'utilisation que l'on veut en faire, sa description sera faite dans les sections suivantes portant sur la spécialisation.

Nous allons maintenant étudier comment un pointeur parcourt un graphe : ses modifications internes, quand il doit se dupliquer . . .

4.1.3 Fonctionnement d'un Pointer

Le fonctionnement d'un pointeur est divisé en deux parties :

- Le parcours du graphe
- Le traitement des noeuds rencontrés

Nous avons déjà vu comment sont constitués les DCxG, nous commencerons donc par voir comment un pointeur peut parcourir un graphe. Dans la seconde partie de cette section, nous verrons comment est constitué un pointeur, les informations qu'il contient et comment il traite chaque nouveau noeud rencontré.

Parcours du graphe Nous allons maintenant voir comment un pointeur parcourt un graphe, pour cela, nous allons nous baser sur l'exemple donné figure 4.4. Cette figure montre le parcours d'un graphe très simple par un pointeur. Le pointeur est représenté par $P(t)$, qui correspond à la position du pointeur P à l'instant t .

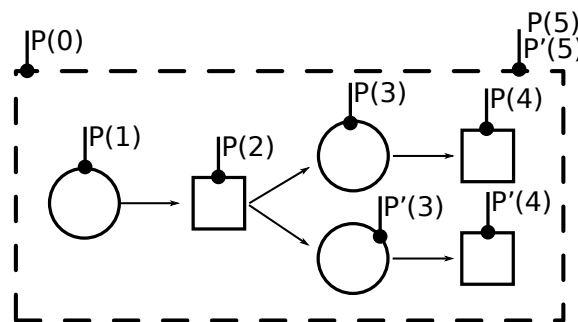


FIG. 4.4 – Parcours d'un pointeur
 $P(t)$ est la position du pointeur à l'instant t .

Lorsqu'un pointeur est créé, on le fait pointer sur le graphe que l'on veut lui faire parcourir, ici $P(0)$.

Lorsqu'il rentre dans un graphe, il va se dupliquer sur le ou les premiers noeuds du graphe, ici : $P(0) \rightarrow P(1)$.

Une fois sur un noeud, le pointeur peut effectuer un traitement. Une fois que ce traitement est fini, le pointeur avance dans le graphe.

Si le pointeur ne possède qu'un seul suivant, alors il se déplace tout simplement : $P(1) \rightarrow P(2)$, $P(3) \rightarrow P(4)$ et $P'(3) \rightarrow P'(4)$.

Si le pointeur possède plusieurs suivants, alors il se duplique sur ses derniers suivants et se déplace sur son premier suivant : $P(2) \rightarrow P'(3)$ et $P(2) \rightarrow P(3)$.

Pour finir si le pointeur n'a pas de suivants, il sort du graphe : $P(4) \rightarrow P(5)$ et $P'(4) \rightarrow P'(5)$.

Une fois arrivé à la fin du graphe, le pointeur se détruit.

L'algorithme 2 reprend les explications précédentes.

Algorithme 2 : Parcours d'un graphe

```
1 switch Type pointé do
2   case Début (sous-)graphe
3     | Nouveaux pointeurs sur les premiers;
4   end
5   case Fin de graphe
6     | Destruction du pointeur;
7   end
8   case Fin de sous-graphe
9     | Reprendre après l'appel du sous-graphe;
10  end
11  case Appel de sous-graphe
12    | Pointer sur le sous-graphe appelé;
13  end
14  case Noeud
15    switch Nombre de suivants do
16      case Pas de suivants
17        | Quitter le graphe;
18      end
19      case Un seul suivant
20        | Avancer sur le prochain noeud;
21      end
22      case Plusieurs suivants
23        | Avancer sur le premier suivant;
24        | Se dupliquer sur les autres suivants;
25      end
26    end
27  end
28 end
```

Pour pouvoir décider quand détruire un pointeur, il faut effectuer des tests lorsque le pointeur pointe sur un certain type de noeud. Arrêter les pointeurs sur chaque noeud complique cette gestion des pointeurs car il faut demander au pointeur d'avancer jusqu'à ce qu'il atteigne un point intéressant du graphe. Les pointeurs vont donc s'arrêter seulement à certains points du graphe, par exemple au début d'un graphe, sur une action, ...

Lors de ces arrêts, on pourra effectuer les tests nécessaire pour décider si le pointeur doit continuer de parcourir la branche du graphe qu'il est en train d'explorer.

On peut stopper un pointeur à trois endroits :

1. Un pointeur peut être stoppé sur un début de graphe. Cet arrêt permettra au gestionnaire associé au pointeur de tester le contexte d'entrée du graphe, pour déterminer la validité du graphe par exemple.
2. Si on demande au pointeur de s'arrêter sur une fin de graphe, le gestionnaire pourra tester le contexte de fin de graphe. Cet arrêt a un intérêt lorsque l'on veut savoir si le graphe a été exécuté avec succès.
3. Nos pointeurs peuvent aussi s'arrêter sur les noeuds d'actions. Le gestionnaire pourra ainsi évaluer la validité de l'action et proposer la meilleure action.

La notion d'arrêt sera développée plus longuement dans les sections suivantes, car ce choix est spécifique à chaque utilisation des pointeurs. Maintenant que nous avons vu comment se déplace un pointeur dans un graphe, nous allons voir comment on peut s'en servir pour extraire des informations.

Traitement des noeuds Pour pouvoir se positionner dans un graphe, un pointeur va disposer de trois attributs :

- Le noeud courant : Permet de savoir où en est le pointeur dans le graphe
- Une pile de graphes : Représente la hiérarchie des graphes et sous-graphes dans lesquels le pointeur est rentré
- Une pile de contextes : Garde une trace des différents contextes rencontrés

L'algorithme 3 présente les modifications internes réalisées par le pointeur à chaque nouveau noeud rencontré. Cet algorithme est exécuté avant chaque passage au suivant. Nous allons maintenant voir comment fonctionne l'algorithme en détail.

Algorithme 3 : Traitement des noeuds

```

1 switch Type pointé do
2   case Début de (sous-)graphe
3     | Empiler le graphe;
4   end
5   case Fin de (sous-)graphe
6     | Dépiler le graphe;
7   end
8   case Contexte
9     | if Précédent était Action then
10    |   | Vider la pile de contexte;
11    | end
12    | Empiler le contexte;
13  end
14  case Action, Appel de sous-graphe
15    | Aucun traitement;
16  end
17 end

```

A chaque noeud contextuel rencontré, le contexte instancié est poussé dans une pile de contextes associée au pointeur. Cette pile de contexte est vidée après chaque action.

Grâce à ce mécanisme, chaque pointeur garde une trace des différents contextes rencontrés.

A chaque début (ou fin) de graphe ou sous-graphe, le pointeur va empiler (respectivement dépiler) le graphe dans sa pile de graphe. Ainsi, le pointeur sait exactement où il se situe dans la hiérarchie du graphe courant.

Note technique : afin de pouvoir reprendre le graphe à la fin d'un appel de sous-graphe, la position dans le graphe appelant est aussi stockée.

Nous disposons dorénavant d'un mécanisme générique que nous allons spécialiser pour les utilisations que nous avons du plan.

4.2 Prise de décision

Dans cette section, nous allons spécialiser le mécanisme décrit précédemment. Le but est de pouvoir donner de l'autonomie à des agents virtuels dans la simulation, afin que ceux-ci puissent prendre des décisions en suivant les stratégies décrites par les DCxG.

Dans la section 4.2.1, nous étudierons comment le gestionnaire doit extraire les données de ses pointeurs.

La section 4.2.2 présente des mécanismes avancés, que nous proposons afin d'améliorer la prise de décision.

4.2.1 Extraction des informations

Dans le cadre de la prise de décision, ce qui va nous intéresser c'est de savoir si un graphe convient bien au contexte actuel et, une fois que le ou les graphes les plus intéressants seront sélectionnés, il faudra trouver la meilleure action à réaliser.

Nous avons donc décidé, de n'arrêter les pointeurs que sur les actions et les début de graphes.

Lors de l'arrêt sur un début de graphe, le pointeur fournira au gestionnaire le contexte d'entrée du graphe. En comparant ce contexte au contexte personnel de l'agent, le gestionnaire pourra déterminer les graphes les plus adaptés. Le gestionnaire ne demandera ensuite qu'aux meilleurs pointeurs de continuer à parcourir le graphe.

Pour ce qui est de savoir si un graphe est adapté ou pas, nous avons décidé de comparer la similitude entre le contexte personnel et le contexte d'entrée à un seuil fixe. En dessous de ce seuil, le pointeur, et donc le graphe pointé seront éliminés. Il ne nous reste ainsi plus que les graphes adaptés à la situation courante.

Ensuite, nous passons au choix d'une action. Pour choisir l'action à réaliser, il faut relancer les pointeurs pour qu'ils puissent s'arrêter cette fois-ci sur une action.

Dans le cas où un pointeur atteindrait un appel de sous-graphe et qu'il s'arrête sur le sous-graphe, il faudrait le relancer jusqu'à ce qu'il atteigne une action. Nous avons décidé en effet de ne pas tenir compte du contexte d'entrée des sous-graphes, nous supposons que les noeuds contextuels précédent l'appel du sous-graphe sont suffisant pour spécifier l'intérêt de la première action du graphe.

Une fois que tous les pointeurs sont positionnés sur une action, on va comparer la pile des contextes avec le contexte personnel. Les résultats obtenus nous permettront de déterminer les meilleures actions à réaliser.

Les pointeurs pointant vers une action dont le score n'est pas maximal seront supprimés. Si plusieurs pointeurs ont le même score, alors tous ces pointeurs restent en course.

Si plusieurs actions ont le même score, nous avons décidé de choisir celle à réaliser au hasard. Nous verrons dans la section suivante une amélioration que nous pouvons apporter.

En répétant l'étape permettant de trouver une action, on va pouvoir trouver la stratégie la plus adaptée aux contextes dans lesquels se trouve l'agent.

Une fois que le gestionnaire n'aura plus de pointeurs, il faudra qu'il en relance de nouveaux sur chaque graphes. En effet, tous les pointeurs ne disparaissent que quand la stratégie est terminée, il est alors temps d'en commencer une autre.

Chaque action renvoyée devra être exécutée par l'agent. Ainsi il réalisera la stratégie la plus adaptée à son contexte personnel.

Nous allons maintenant voir deux mécanismes permettant d'améliorer la sélection d'action.

4.2.2 Fonctionnement avancé

Choix entre plusieurs actions Si plusieurs stratégies nous sont proposées avec un même score, il va falloir faire un choix. La méthode la plus simple est de tirer au hasard, cependant ce n'est pas forcément celle qui fournira les meilleurs résultats.

La méthode proposée par la suite va nous permettre d'exploiter les informations données par le contexte de fin de graphe, elle est inspirée de l'algorithme proposé par M. Bauer [1]. L'idée est simple, on a plus intérêt à utiliser une stratégie qui a bien fonctionné qu'une stratégie qui ne marche pas bien.

Ainsi à chaque fin de graphe, on va calculer la similitude entre le contexte personnel et celui de fin. Le résultat nous permettra pondérer le graphe en fonction. Si le graphe a un bon résultat on donne plus de poids au graphe et si le graphe a un mauvais résultat on fait descendre le poids.

Lors du tirage au sort, on associera des probabilités plus fortes aux graphes les plus performants, tout en gardant une chance d'exécuter d'autres graphes, afin de ne pas bloquer le système toujours sur le même graphe.

Gestion des contextes d'exceptions Lorsque l'on applique une stratégie dans un environnement dynamique, il est possible qu'un élément extérieur vienne à interrompre la stratégie. Pour cela nous avons décidé de créer les contextes d'exceptions.

Pour une utilisation efficace, les contextes d'exceptions doivent être vérifiés avant de chercher la meilleure action. Si le contexte d'exception dépasse un seuil fixé, alors il faut détruire le pointeur pour essayer de libérer les graphes et trouver une nouvelle stratégie à appliquer.

4.3 Assistance pédagogique

Le but de cette section va être de spécialiser le mécanisme générique présenté précédemment dans le but d'apporter une assistance pédagogique à l'apprenant.

Pour assister l'apprenant, nous avons besoin de connaître la stratégie qu'il est en train d'appliquer. Il nous faudra comparer les actions qu'il réalise aux stratégies connues.

Cette fois encore, ce sont un gestionnaire et des pointeurs qui vont nous permettre d'effectuer cette tâche. Nous étudierons donc dans un premier temps la spécialisation du gestionnaire et des pointeurs.

Par la suite, nous verrons comment nous pouvons utiliser les données renvoyées pour apporter deux types d'aide pédagogiques.

Pour finir nous proposerons des mécanismes avancés pour apporter une meilleure assistance à l'apprenant.

4.3.1 Spécialisation

Pour pouvoir effectuer le suivi des actions de l'apprenant nous ne pouvons nous baser que sur les actions qu'il a réalisées. Nos pointeurs ne vont donc s'arrêter que sur les actions.

Une fois l'action réalisée, on supprimera tous les pointeurs ne pointant pas sur l'action réalisée par l'apprenant. A chaque nouvelle action réalisée, on fera avancer les pointeurs restant sur une nouvelle action.

Ainsi, au bout d'un petit nombre d'actions, il ne devrait plus rester que les pointeurs concernant les stratégies que peut être en train d'appliquer l'apprenant.

Le contexte stocké dans la pile des contextes pourra être comparé avec le contexte personnel de l'apprenant. La différence entre ces contextes nous permettra de déterminer si l'apprenant a réalisé une action adéquate ou pas.

Quand le gestionnaire n'aura plus de pointeurs, il en re-crèera sur tous les graphes afin de trouver la nouvelle stratégie commencée.

Cette absence de pointeur peut arriver si l'apprenant change de stratégie ou s'il a fini une stratégie.

4.3.2 Deux types d'aides

Grâce au suivi d'action que nous venons de définir, nous allons pouvoir apporter deux types d'aides détaillées dans [3].

Le premier type permet un guidage fort qui se produit avant l'action et le second permet de mettre en évidence les éléments pertinents après l'action, le guidage est alors moins fort.

Dans le premier type d'aide, celle-ci est effectuée avant que l'apprenant n'agisse dans la simulation. Cette aide repose principalement sur la prédiction de l'action à effectuer. Elle peut être mise en place grâce au suivi de graphe contextuel qui nous renseigne sur la stratégie en cours.

Ainsi il est possible de représenter dans l'environnement les éléments pertinents qui vont permettre à l'apprenant de mieux choisir son action. Éventuellement, on pourra aussi représenter directement l'action à réaliser par une primitive graphique.

Le deuxième type d'aide repose lui aussi sur les graphes contextuels, mais plus particulièrement sur les noeuds contextuels. On s'intéresse alors aux éléments que l'apprenant pourrait avoir oubliés de prendre.

Pour déterminer ces éléments, nous nous basons sur la différence entre le contexte courant et le contexte attendu. L'absence ou les changements de valeur des perceptions nous permettent de déterminer celles qui n'ont pas été prises en compte.

Il est alors possible de les mettre en valeur dans la simulation. Cette aide est possible grâce à des primitives graphiques que l'on associe à chaque perception.

Un exemple présenté dans le chapitre 6 montre la mise en place d'une aide mettant en évidence les raisons permettant de justifier une action.

4.3.3 Fonctionnement avancé

Actions cycliques Comme nous nous trouvons dans un environnement dynamique, nous avons introduit les actions cycliques. Il va donc nous falloir traiter ce genre d'action, dans le cas où l'apprenant réalise plusieurs fois la même action.

Pour permettre la reconnaissance des actions cycliques, nous proposons de considérer l'action cyclique comme une action bouclant sur elle-même. Ainsi lorsqu'un pointeur va vouloir continuer après l'exécution d'une action cyclique, il va retourner sur l'action et sur va se dupliquer sur les suivants de cette action.

Ainsi tant que l'action est répétée, les pointeurs pointant sur les suivants seront détruits, mais lorsqu'un suivant sera réalisé, ce sera le pointeur de l'action cyclique qui sera supprimé.

La figure 4.5 représente le suivi d'un action répétée deux fois.

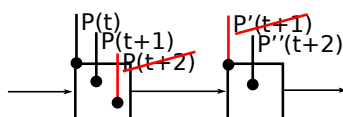


FIG. 4.5 – Pointeurs suivant une action cyclique

Nous admettons ici que l'action cyclique sur laquelle pointe P à l'instant t est réalisée une première fois. A l'instant $t + 1$, le pointeur va se dupliquer sur son suivant et retourner sur l'action.

Vu que l'on refait l'action une deuxième fois, $P'(t + 1)$ va être détruit. Le pointeur $P(t + 1)$ va de nouveau se dupliquer sur son suivant et retourner sur l'action.

Cette fois si c'est l'action suivante qui va être réalisée. Ce sera alors $P(t + 2)$ qui sera supprimé et $P''(t + 2)$ qui continuera le suivi des actions de l'apprenant.

Action parasite Il se peut que l'apprenant exécute des actions différentes de celles proposées par le graphe, par exemple des actions parasites. Dans ce cas, il faudrait que le pointeur ne soit pas détruit pour que l'on puisse continuer de suivre le graphe.

La solution proposée est de ne pas systématiquement supprimer le pointeur afin de "lui donner une chance".

Pour déterminer quand le supprimer, nous proposons deux solutions :

- L'une se basant sur le calcul de la similitude entre la pile des contextes et le contexte personnel : Si cette similitude est élevée, c'est que l'action pointée est très pertinente, le pointeur sera donc utile pour plus tard. Si la similitude est faible, alors l'action n'est pas très pertinente, on peut donc détruire le pointeur.
- L'autre se basant sur un délai : On va permettre au pointeur de subir un certain nombre d'échecs. Au delà, on considérera que l'action n'était vraiment pas la bonne et on détruira le pointeur.

Chapitre 5

Apprentissage rapide de comportements

Nous l'avons vu précédemment, les DCxG peuvent être définis par un formateur manuellement, grâce à leur représentation graphique. En nous inspirant de la programmation par démonstration, nous avons aussi voulu proposer un algorithme capable d'apprendre des DCxG à partir d'exemples réalisés dans l'environnement virtuel par un humain.

Nous avons décidé de nous inspirer de la programmation par démonstration car elle est capable d'effectuer un apprentissage à partir d'un petit nombre d'exemples.

Du fait de la structure des DCxG, notre algorithme ne pouvait pas se baser sur :

- L'apprentissage de règle vu dans FOIL [15, 17], car ces règles ne prennent pas en compte le contexte mais seulement les actions.
- L'apprentissage de modèles non déterministes comme SHEEPDOG [13], car notre modèle est déterministe.
- L'apprentissage de séquences de couple état-action utilisé par COMETS [11] nous a semblé insuffisant pour pouvoir apporter un réel support pédagogique par la suite.
- L'algorithme utilisé par ACTIONSTREAMS [16] qui, pour sa part, ne prend pas en compte les états successivement rencontrés.

S'il nous était impossible d'utiliser un des algorithmes ci dessus, nous avons proposé notre propre algorithme qui, dans certaines parties, s'inspire tout de même des travaux effectués dans le domaine de la programmation par démonstration.

Après avoir vu les données brutes que nous récupérerons de la simulation, nous verrons chacun des points suivants, répondant chacun à un besoin de notre algorithme :

- Filtrage des données brutes
- Détection des points clés des exemples
- Création de branchements conditionnels

5.1 Données pour l'apprentissage

Comme le présente la figure 5.1, les seules données que nous allons pouvoir récupérer en provenance de la simulation sont le contexte personnel de la personne observé, et les actions qu'elle a réalisé.

Dans notre cas, rappelons que nous considérons le contexte personnel comme l'ensemble des perceptions de l'avatar dans la simulation.

Ne possédant que ces deux informations, notre base d'apprentissage va être une succession de contextes et d'actions. Il pourra éventuellement y avoir plusieurs contextes entre deux actions et inversement, plusieurs actions successives sans nouveau contexte.

Par la suite nous appellerons une séquence de contextes et d'actions un scénario.

Afin de simplifier la phase d'apprentissage, nous nous basons sur un ensemble d'exercice ayant le même but. Le début et la fin d'un scénario sont déterminés par l'utilisateur.

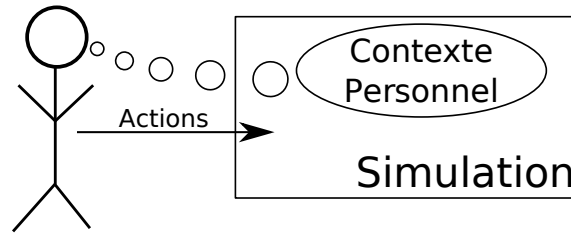


FIG. 5.1 – Echanges entre la simulation et l’algorithme d’apprentissage

Nous allons maintenant apporter quelques précisions sur les scénarios.

Si nous disposons bien d’une notion de succession entre les différentes actions, nous ne possédons pas de notion de temps. En effet si l’agent n’effectue pas d’actions pendant un certain temps, aucune action ne va être enregistrée dans le scénario.

Il est donc préconisé de prévoir une action “attendre” permettant de modéliser cette attente.

L’insertion d’un nouveau contexte peut être effectuée de deux façons.

La première consiste à insérer le contexte personnel de l’agent de manière périodique. L’avantage de cette solution est qu’on est sûr de pouvoir suivre l’évolution du contexte. Cependant la mise à jour du contexte peut intervenir après l’exécution d’une action, alors que le contexte aura été pertinent pour l’action.

La seconde solution consiste à insérer le contexte avant chaque début d’action. L’avantage cette fois est que nous aurons toujours le contexte correspondant au moment où l’action a été exécutée. Cette fois-ci ce sont des contextes intermédiaires qui risquent de nous manquer.

La solution idéale consiste alors à utiliser une combinaison des deux méthodes : un mécanisme insérant périodiquement le contexte personnel de l’agent et l’exécution d’une action provoquant elle aussi l’insertion du contexte personnel de l’agent. Ensuite on filtrera les contextes successifs pour détecter les changements de contextes.

Nous verrons dans la section suivante comment notre algorithme est capable de réduire le nombre de contextes insérés.

Pour finir nous allons présenter un exemple simplifié de ce que nous pouvons récupérer. La figure 5.2 présente un exemple de scénarios représentant des actions réalisées dans un exercice de football. Les points représentent les joueurs perçus. L’apprenant est dans l’équipe rouge et a la balle.

Dans ces exemples, l’apprenant réalise deux types d’actions :

- Avancer vers le but
- Faire une passe à A

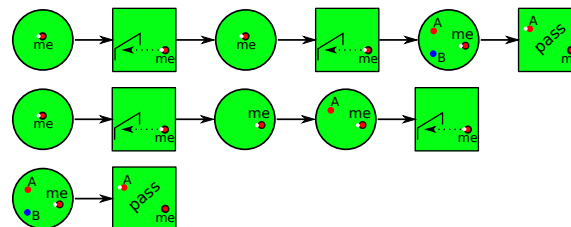


FIG. 5.2 – Un exemple de scénarios

5.2 Filtrage des scénarios

Le filtrage des scénarios est la première étape lors de l’apprentissage. Son déroulement peut être découpé en plusieurs phases :

1. Élimination des contextes superflus.
2. Pondération des perceptions en fonction de leur importances.

Chacune des phases mentionnées ci dessus fera l'objet d'une des parties suivantes.

Par la suite nous étudierons deux mécanismes avancés, restant encore à définir afin d'obtenir des plans plus performants :

1. Regroupement des actions cycliques.
2. Remplacement d'une portion de scénario par un l'appel d'un graphe existant.

5.2.1 Filtrage des contextes

Lorsque l'on utilise la méthode présentée dans la section précédente pour l'insertion des contextes personnels, il se peut que l'on insère plusieurs fois de suite un même contexte.

Le but de cette phase va être d'éliminer les contextes successifs redondants.

Le but est très simple, on se base sur le calcul de la similitude entre deux contextes pour déterminer si deux contextes sont très similaires. Si la similitude est supérieure à un seuil fixé, alors on supprime le contexte précédent.

Nous avons fait le choix de supprimer le contexte précédent car c'est le contexte qui est le plus loin de l'action suivante, donc le moins approprié pour la prise de décision.

L'algorithme 4 retranscrit cette phase.

Algorithme 4 : Élimination des contextes superflus dans un scénario

```

1 foreach Noeud du scénario do
2   if Noeud de type contexte then
3     if Précédent de type contexte then
4       Comparer(Noeud courant, Noeud précédent);
5       if Similitude > seuil then
6         | Supprimer(Noeud précédent);
7       end
8     end
9   end
10 end

```

5.2.2 Filtrage des perceptions

Une fois que nous avons filtré tous les contextes, nous allons vouloir éliminer les perceptions parasites n'ayant pas à voir avec la stratégie à apprendre.

En effet, dans le scénario, **toutes** les perceptions perçues par l'agent sont ajoutées, nous allons essayer de déterminer quelle sont les perceptions pertinentes pour chaque contexte.

Pour réaliser ce filtre, nous proposons plusieurs heuristiques qui vont nous permettre de pondérer les perceptions :

1. Toute perception dont la valeur a changé après l'exécution de l'action doit avoir de l'importance pour l'action.
2. Un paramètre qui varie pour différents types d'action n'est certainement pas spécifique à une action et n'est donc pas important pour la prise de décision.
3. Un même paramètre qui change à chaque fois qu'un type d'action est réalisée est pertinent pour ce type d'action.

En appliquant ces heuristiques sur l'ensemble des scénarios, nous espérons pouvoir filtrer avec une assez bonne qualité les perceptions.

5.2.3 Reconnaissance d'actions cycliques

Nous travaillons dans un environnement dynamique, nous savons donc que nous allons avoir des actions répétées. Il sera donc nécessaire dans les prochaines versions de l'algorithme de pouvoir remplacer la répétition d'une même action par une action cyclique.

Une action cyclique peut être représentée d'au moins deux manières :

- Attendre que le contexte précédent soit vrai
- Exécuter tant que le contexte précédent est vrai

La détection de la répétition d'une action est facile, cependant la détection de la condition d'arrêt reste un mécanisme complexe.

Cette détection demandant à notre sens trop de travail, nous avons décidé de laisser cette étude pour plus tard. Cette étude sera certainement l'occasion de définir de nouvelles conditions d'arrêt.

5.2.4 Reconnaissance de sous-graphes

Le but de cette phase est de remplacer des portions de graphe par l'appel d'un sous-graphe existant.

L'étude de ce mécanisme a été laissée pour un travail ultérieur, cependant nous proposons une piste : nous disposons d'un mécanisme capable de réaliser le suivi des actions d'un utilisateur. En modifiant ce mécanisme, nous pouvons l'utiliser pour détecter l'utilisation d'un graphe déjà existant. Il suffira alors de remplacer la portion identifiée par un appel au sous-graphe correspondant.

5.3 Recherche des situations clés

Une fois que nous avons filtré les scénarios, nous allons essayer d'en trouver les situations clés. En effet notre hypothèse est que les exercices peuvent être découpés en plusieurs sous-séquences permettant chacune d'atteindre une situation. Le passage par toutes ces situations permet la réussite de la stratégie.

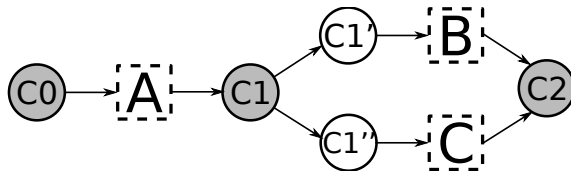


FIG. 5.3 – Les scénarios possèdent des points clés communs

Dans l'exemple donné figure 5.3, C_0 , C_1 et C_2 sont des points clés alors que C_1' et C_1'' sont des points de divergence.

La sous-séquence A est une manière d'aller de C_0 à C_1 . Les séquences B et C sont deux manières d'aller de C_1 à C_2 . B est à utiliser dans le contexte $C_1 + C_1'$ alors que C sera à utiliser dans le contexte $C_1 + C_1''$.

Cette section se consacre donc à la recherche des points clés.

Dans un premier temps nous verrons comment nous allons arriver à aligner les scénarios pour ensuite permettre la seconde étape consistant à identifier les points clés.

5.3.1 Alignement des scénarios

Pour aligner nos scénarios nous utilisons une méthode algorithmique appelée programmation dynamique. Elle va nous permettre de faire correspondre les noeuds de deux scénarios, on dit alors qu'ils sont alignés.

Grâce à cet algorithme, les contextes les plus similaires vont se retrouver alignés l'un avec l'autre.

L'algorithme utilisé ne permet l'alignement que de deux scénarios. Pour pouvoir aligner tous les graphes entre eux, et comme notre apprentissage est réalisé à partir d'un petit nombre d'apprentissage, nous proposons d'utiliser l'algorithme 5.

Algorithme 5 : Alignement de tous les scénarios

```
1 foreach Scénario do  
2 | Aligner avec le scénario suivant();  
3 end  
4 foreach Scénario do  
5 | Aligner avec le dernier scénario();  
6 end
```

A la fin de la première boucle, chaque scénario n sera aligné avec les $n - 1$ scénarios le précédent. Le dernier scénario sera donc aligné avec tous les scénarios.

Nous ré-alignons donc tous les graphes avec le dernier pour que tous les scénarios soient alignés entre eux.

A partir des scénarios présentés à la figure 5.2, nous nous arrivons à l'alignement de la figure 5.4.

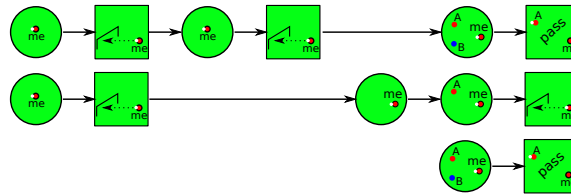


FIG. 5.4 – Les scénarios possèdent des points clés communs

5.3.2 Identification de points clés

Maintenant que tous nos scénarios sont alignés, nos contextes clés devraient se trouver à la même position.

Pour trouver les points clés, nous allons donc passer en revue toutes les positions des séquences que nous avons. Si à une même position, toutes les séquences ont un contexte ayant tous une similitude élevée, alors nous avons trouvé un point clé.

L'intérêt d'avoir pondéré les perceptions est que nous allons pouvoir effectuer nos calculs de similitude en tenant compte en priorité des perceptions pertinentes.

Le seuil de similitude entre les différents contextes ne doit cependant pas avoir une valeur élevée car, comme nous allons le voir des contextes clé peuvent posséder des petites différences, notamment lorsque l'on rencontre un point de choix. La figure 5.5 représente un exemple correspondant au résultat donné figure 5.3.

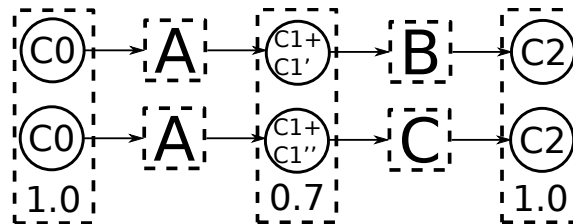


FIG. 5.5 – Les points clés ne sont pas forcément identiques

5.4 Génération du graphe

Cette dernière étape va permettre, une fois les situations clés identifiées de générer un graphe. Cela revient à passer de la figure 5.5 à la figure 5.3.

Ce résultat est obtenu en plusieurs étapes que nous allons maintenant détailler.

Tout d'abord, nous allons tester si les sous-séquences entre les points clés sont identiques ou pas.

Nous proposons une manière simple de tester la similarité entre deux séquences : l'utilisation du suivi d'action (cf. section 4.3).

Nous supposons en effet que c'est grâce à l'action que l'on obtient la situation clé suivante. Si les actions sont identiques, alors on est face à la même manière d'arriver au résultat.

Si deux séquences sont identiques alors nous en supprimons une et nous utilisons l'intersection de leur contextes pour définir le contexte nécessaire pour utiliser cette sous-séquence. En effet, si les sous-séquences sont identiques, c'est quelles nécessitent le même contexte pour être utilisées.

Dans le cas de deux sous-séquences différentes, aucun regroupement ne sera fait.

Une fois que nous avons trié les sous-séquences, nous allons retenir l'intersection de tous les contextes d'entrée comme le contexte principal ($C1$ dans l'exemple figure 5.3). Les différences entre les contextes permettront de définir le contexte spécifique pour chaque sous-séquence ($C1'$ et $C1''$ dans l'exemple figure 5.3).

En utilisant les mécanismes que nous venons de voir, nous ne pouvons obtenir des graphes qu'avec un niveau de choix. Il pourrait être intéressant d'exécuter récursivement les procédures d'alignement et de regroupement pour peut être trouver des points de choix à l'intérieur de points de choix.

Cette piste sera à poursuivre dans un travail ultérieur.

Chapitre 6

Résultats

Dans les chapitres précédents, nous avons pu étudier le modèle que nous proposons dans le cadre de la modélisation de comportements non-procéduraux. Les graphes contextuels dynamiques (DCxG) sont basés sur les graphes contextuels définis par Brézillon dans [7]. Nous les avons étendu pour pouvoir prendre en compte l’aspect dynamique de notre environnement.

Au delà de la proposition d’un modèle, nous nous sommes efforcés de fournir un ensemble de mécanismes permettant l’exploitation des DCxG.

Ainsi, dans le chapitre 4, nous avons proposé des mécanismes permettant la prise de décision et l’assistance pédagogique.

Finalement, dans le chapitre précédent, nous avons vu une première approche de l’apprentissage des DCxG.

Le but de ce chapitre va être de présenter ce que nous avons pu mettre en pratique et les résultats que nous avons obtenus.

Dans un premier temps nous verrons une présentation de la librairie développée durant ce stage de Master. Cette librairie, nommée libArCAS, permet la gestion des DCxG, leur utilisation pour l’exécution et une première approche d’assistance pédagogique et d’apprentissage.

Dans un second temps, nous allons présenter l’intégration de notre librairie dans une simulation de foot. Cette section sera agrémentée de plusieurs captures d’écran.

La section suivante traitera de deux problèmes que nous avons rencontré lors de la mise en place de la libArCAS dans la simulation de foot :

- Le premier a été de récupérer un paramètre du contexte pour pouvoir faire agir dessus.
- Le second est un problème lié à la perception mise en oeuvre dans la simulation de foot.

Ensuite, nous présenterons nos résultats pour l’apprentissage de stratégies. Nous verrons alors des exemples d’alignement de scénarios.

Pour finir, nous aborderons le problème de la collaboration que nous n’avons pas traité dans les DCxG et qui pourra faire l’objet d’un travail ultérieur.

6.1 Implémentation

Les graphes contextuels et les mécanismes associés ont été implémentés sous forme d’une bibliothèque en C++, en utilisant ARéVi¹. Cette librairie sera nommée par la suite libArCAS pour “ARéVi Context and Action System”.

Nous allons maintenant voir les différentes parties qui composent la librairie.

6.1.1 Gestion des DCxG

La figure 6.1 représente l’organisation de nos différentes classes permettant la gestion des DCxG.

¹Atelier de Réalité Virtuelle, développé au CERV

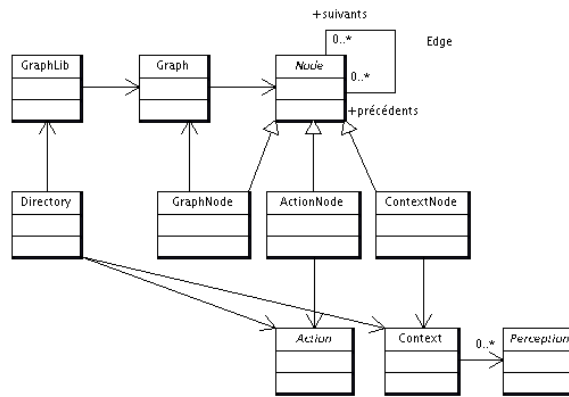


FIG. 6.1 – Les classes permettant la gestion des DCxG

La classe `DIRECTORY` sert d'annuaire et va référencer toutes les bibliothèques de graphes, les actions et les contextes.

Les classes `PERCEPTION` et `ACTION` sont à sur-définir dans la simulation. Elles permettent de décrire ce que perçoit l'agent et ce qu'il peut faire.

Une bibliothèque de graphes (`GRAPHLIB`) va être composée de graphes (`GRAPH`). Chaque graphe sera lui même composé de noeuds.

Les noeuds peuvent être instanciés sous trois formes différentes :

- `CONTEXTNODE` : un noeud instanciant un contexte.
- `ACTIONNODE` : un noeud instanciant une action.
- `GRAPHNODE` : un noeud représentant un appel de graphe.

Au point de vue méthode, la plus importante est celle implémentée dans la classe `DIRECTORY` qui permet le chargement de bibliothèques de DCxG à partir d'un fichier XML (cf. FIG. 6.2).

6.1.2 Gestion des pointeurs

La figure 6.3 représente nos classes permettant le parcours des graphes par des pointeurs.

Les instances de `POINTER` et `MANAGER` sont toutes référencées dans `DIRECTORY`. C'est `DIRECTORY` qui va permettre de connaître tous les `POINTER` associés à un `MANAGER`.

Les classes `POINTER` et `MANAGER` implémentent les mécanismes génériques permettant le parcours de graphes. Cependant ces classes sont abstraites et il faudra utiliser les classes `EXECUTOR` et `WATCHER` pour pouvoir utiliser les DCxG dans la simulation.

Les classes `POINTEREXECUTOR` et `POINTERWATCHER` sont des spécialisations des pointeurs spécifiant quand doit s'arrêter le pointeur.

Du point de vue des méthodes, la plus intéressante est la méthode `nextStep()` de la classe `POINTER` qui fait avancer le pointeur jusqu'à ce qu'il soit sur un noeud sur lequel il doit s'arrêter.

Cette méthode est suffisamment paramétrable pour ne pas avoir à être sur-définie dans les classes `POINTEREXECUTOR` et `POINTERWATCHER`.

6.2 Utilisation dans un exercice de foot

La simulation de foot est une représentation d'un terrain avec ses buts, de plusieurs joueurs répartis en deux équipes et d'un ballon (cf FIG. 6.4).

Dans la simulation nous allons disposer d'éléments perçus par l'agent (de type `CONTEXTUALIZEDOBJECT3D`) et d'autres éléments qui ne seront pas pris en compte. L'agent perçoit seulement les buts, le ballon et les joueurs.

Les `CONTEXTUALIZEDOBJECT3D` dans le champ de vision de l'agent vont permettre de générer l'ensemble des perceptions formant le contexte personnel de l'agent (cf. FIG. 6.5).

```

<ArCAS>
  <Context id="C0">
    <PerceptionType source="..." target="..." value="..."/>
  </Context>
  <GraphLib id="lib">
    <Graph id="exemple1">
      <ContextNode id="node0" ref="C0"/>
      <ActionNode id="node1">
        <Action type="MyType1" id="A"/>
      </ActionNode>
      <ContextNode id="node2">
        <Context id="C1">
          <PerceptionType source="..." target="..." value="..."/>
        </Context>
      </ContextNode>
      <ActionNode id="node3">
        <Action type="MyType2" id="B"/>
      </ActionNode>
      <Edge from="node0" to="node1"/>
      <Edge from="node1" to="node2"/>
      <Edge from="node2" to="node3"/>
    </Graph>
  </GraphLib>
</ArCAS>

```

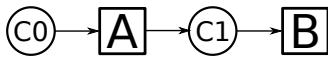


FIG. 6.2 – Un exemple de fichier XML et son résultat

Nous avons défini plusieurs types de perceptions nécessaires pour écrire nos premiers graphes, la liste suivante reste à compléter :

- HasBall : permet de savoir si un joueur possède la balle (vrai ou faux)
- Distance : permet de connaître la distance entre deux CONTEXTUALIZEDOBJECT3D (de 0 pour très près à 4 pour loin)
- Partner : permet de désigner les coéquipiers et les adversaires (vrai ou faux)

Nous avons aussi défini plusieurs actions que l'agent peut réaliser :

- Turn : tourner à droite ou à gauche
- GoToward : avancer tout droit ou vers un CONTEXTUALIZEDOBJECT3D
- Shoot : tirer dans le ballon tout droit ou vers un CONTEXTUALIZEDOBJECT3D
- Pass : faire la passe à un coéquipier
- Call : appel de balle à un coéquipier

Grâce à ces sur-définitions de PERCEPTION et ACTION, nous avons pu décrire nos premiers graphes, en association avec des psychologues de l'équipe ASAP qui nous ont proposé une stratégie de "passe et va" que nous allons voir maintenant.

6.2.1 Exécution de graphes

Nous allons maintenant voir comment nous avons modélisé la stratégie de "passe et va" (cf. FIG. 6.6) qui permet de passer un adversaire à l'aide d'un coéquipier.

Pour une meilleure visibilité, les figures ont été refaites avec un logiciel de dessin vectoriel mais une vidéo est disponible.

La situation initiale est la suivante :

- Le joueur rouge le plus avancé (A) possède la balle et va vers le but.
- Le coéquipier en retrait (B), s'aperçoit que le joueur A va être en difficulté car un adversaire (C) s'approche de lui.

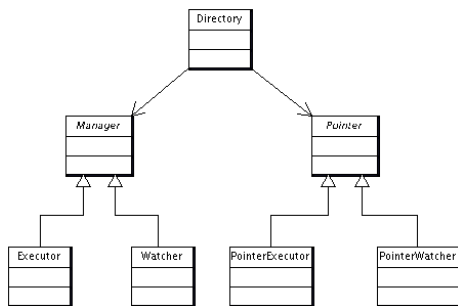


FIG. 6.3 – Les classes permettant la gestion des DCxG

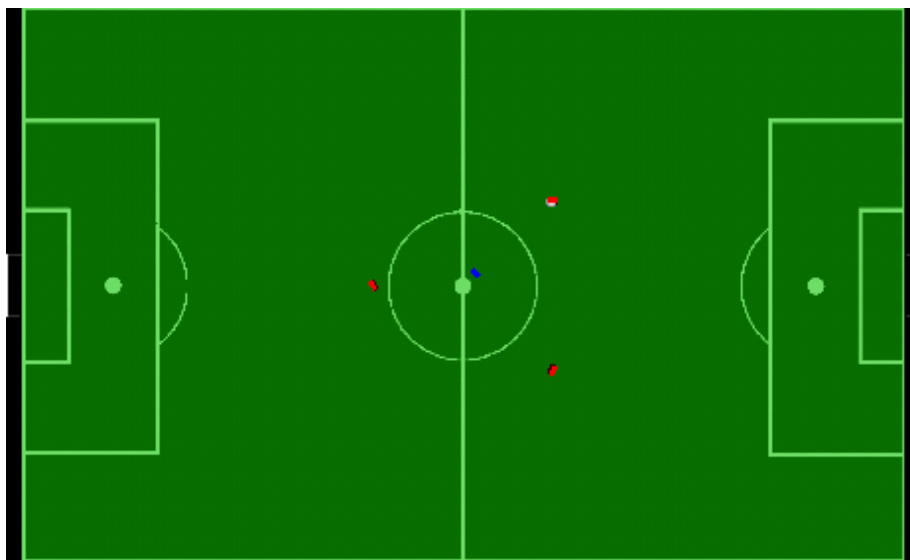


FIG. 6.4 – Un exercice de passe

Le joueur bleu doit attraper la balle, les joueurs rouges n'ont pas le droit de bouger mais peuvent faire des passes.

Il décide alors de provoquer une situation de “passe et va”.

Pour provoquer le “passe et va”, B va accélérer et dépasser A puis il fait un appel de balle pour dégager A. A fait la passe.

Ensuite, A va dépasser C, puis faire un appel de balle.

Pour finir, A tire et marque un but. On pourra remarquer que C n'inquiète pas A.

6.2.2 Assistance pédagogique

Le temps demandé pour développer la gestion des graphes, l'intégration dans l'exercice de foot et une première ébauche de l'apprentissage de graphe ne nous a pas permis d'avoir une vraie interface d'aide à l'apprentissage.

Cependant nous avons pu développer une extension de l'exécution de graphe qui met en évidence les perceptions qui ont permis la prise de décision.

Pour montrer ce que peut donner l'assistance pédagogique, nous avons ajouté un algorithme permettant de montrer pourquoi l'agent prend une décision.

Dans l'exemple du passe et va, nous allons obtenir la mise en évidence de l'adversaire (cf. FIG. 6.7). En effet c'est à cause de l'adversaire que le joueur en retrait va décider de provoquer le “passe et va”.

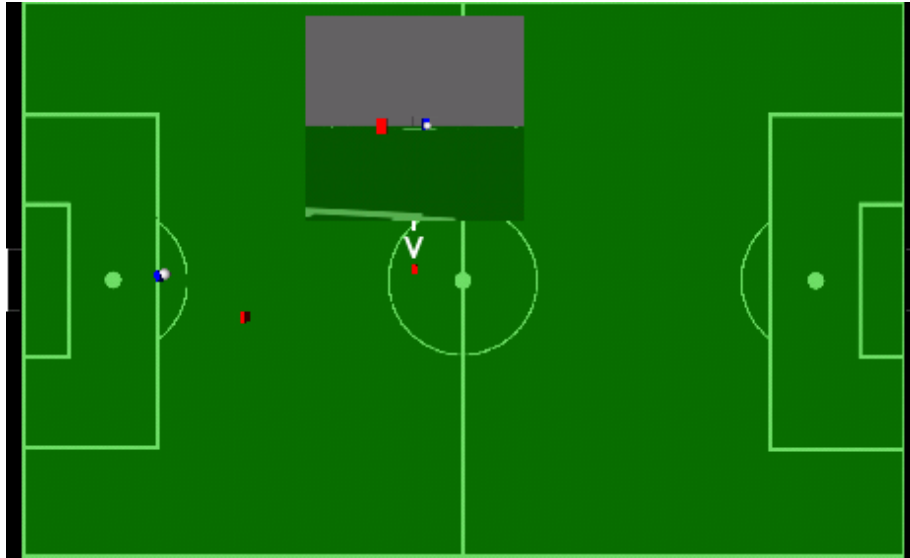


FIG. 6.5 – Ce que perçoit un agent dans la simulation

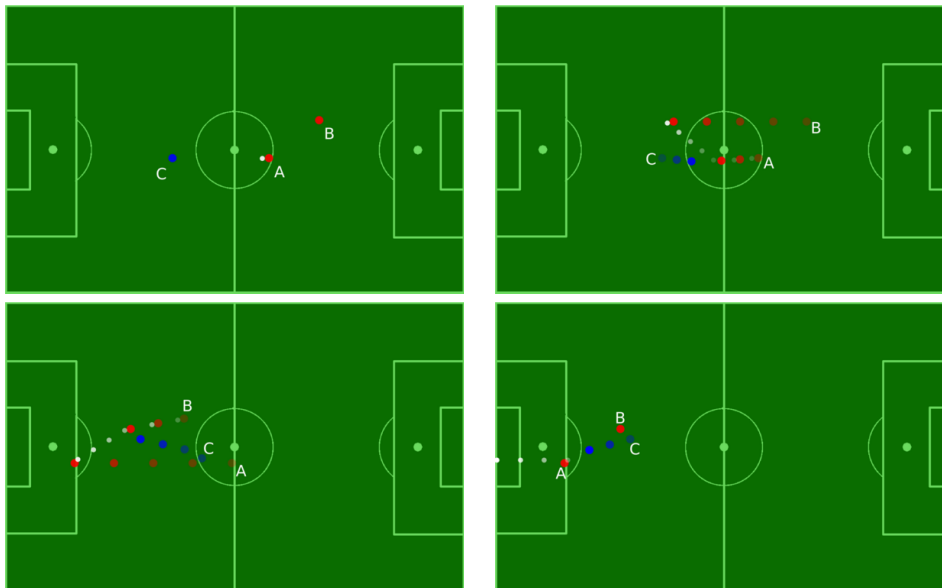


FIG. 6.6 – Exercice de passe et va

6.3 Unification et mémoire

Lorsque nous avons commencé à décrire nos premiers contextes, nous avons réalisé que lorsque l'on désignait un joueur proche, on ne spécifiait pas qui était le joueur.

Le principal problème est arrivé quand nous voulions faire une passe à ce joueur car nous n'avions aucune référence à ce joueur, comment le retrouver ?

Il était impossible de demander à l'action de ré-interpréter les perceptions pour retrouver le joueur. Nous avons donc réalisé un système d'unification capable de trouver remplacer une inconnue par sa valeur et de retourner la valeur dans l'action.

Nous avons développé un système d'unification, permettant l'utilisation de variables et capable d'attribuer une valeur à une variable lors de la comparaison.

Par exemple, dans l'exemple ci-dessous, l'unification permet d'associer la valeur b à X :

– $distance(a, b, proche)$

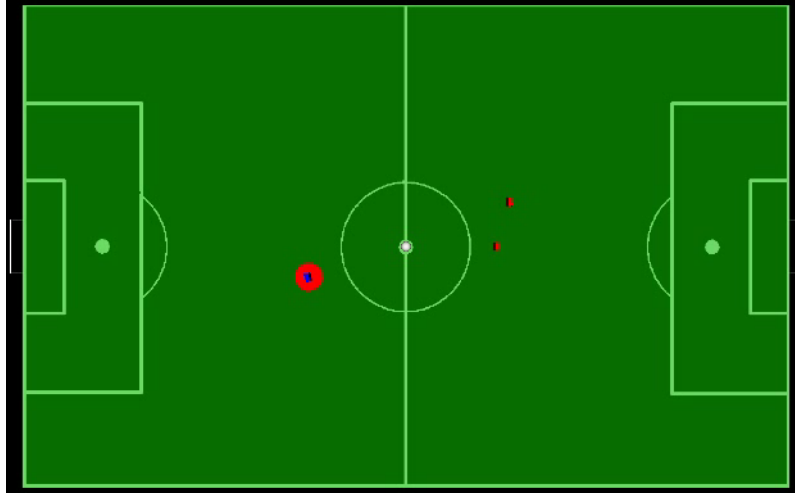


FIG. 6.7 – L’adversaire en sur-brillance incite le joueur en retrait à provoquer un “passe et va”

– $distance(a, X, loin)$

Grâce au système d’unification, nous avons pu faire correspondre la variable A dans la perception $distance(me, X, proche)$ et dans l’action $passe(X)$. Ainsi lorsque le joueur A est proche de moi je peux lui faire une passe.

Lors du calcul de la similitude entre deux contextes, on associera aux variables les valeurs donnant la meilleure similitude.

Lorsque la variable est utilisée uniquement dans un contexte et une action, tout va bien, seulement nous décrivons des stratégies et il est possible que la variable soit utilisée plusieurs fois dans la stratégie.

Dans ce cas nous avons du utiliser un mécanisme de mémoire pour que la même variable ai la même valeur tout le long du graphe.

La mémoire présente cependant un problème, une fois unifiée, la variable ne peut pas changer de valeur. Si on unifie dès le début avec une mauvaise valeur alors la stratégie risque de mal s’exécuter.

Lors des prochaines comparaisons, si $X = b$, $distance(a, X, loin)$ ne pourra plus être comparé qu’à des $distance(a, b, ...)$.

6.4 Collaboration

Grâce au mécanisme d’unification que nous venons de voir, nous pouvons, lors de la comparaison de notre contexte personnel avec un contexte décisionnel, trouver le joueur correspondant le mieux au contexte.

Cependant, lorsque nous réalisons une stratégie à deux, par exemple dans le cas du “passe et va”, deux graphes sont nécessaires :

- Un graphe pour le joueur initiant la stratégie
- Un graphe pour le coéquipier afin de lui permettre de bien réagir à la stratégie

Dans l’état actuel de notre étude, nous ne pouvons pas représenter ce mode de collaboration. Nous avons défini les deux graphes dans deux bibliothèques différentes et attribué une bibliothèque à chaque joueur.

Ainsi il nous manque un mécanisme qui permettrait aux joueur de s’échanger des messages pour permettre de décider de l’application d’une stratégie. Dans un match avec deux équipes complètes, on pourra ainsi laisser le joueur possédant la balle décider avec qui il va faire le “passe et va”.

De même, la collaboration pourrait permettre d’unifier les variables au début d’une collaboration aux bonnes valeurs.

6.5 Apprentissage

Nous n'avons pas eu le temps de développer l'algorithme complet permettant l'apprentissage. Nous allons ici présenter nos résultats concernant l'alignement de scénarios et la détection de points clés.

6.5.1 Limites de l'alignement

Pour les tests réalisés sur la procédure d'alignement, nous avons repris l'exemple du bateau qui évite des rochers. La pointe blanche représente le bout du bateau et les tâches grises des rochers.

La figure 6.8 représente dans sa partie de gauche : trois scénarios non alignés et dans sa partie droite : les trois scénarios alignés.

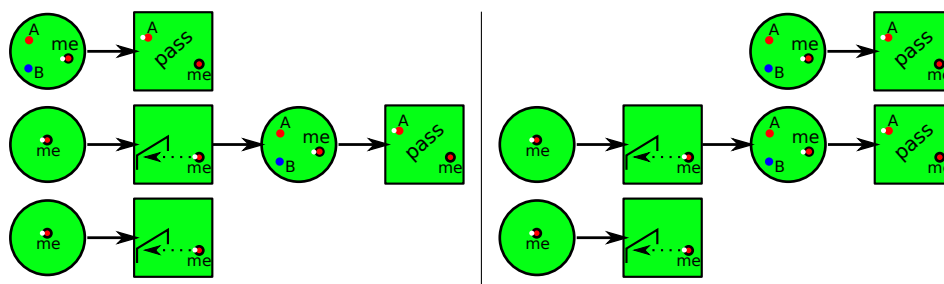


FIG. 6.8 – Alignement de 3 scénarios
A gauche les scénarios non alignés et à droite le résultat de l'alignement

L'algorithme aligne bien les contextes et les actions ensemble.

Un petit problème subsiste cependant. L'alignement ne peut pas insérer d'espaces avant le premier noeud. Pour résoudre ce problème nous avons dupliqué le premier contexte pour que l'alignement puisse marcher.

6.5.2 Un exemple complet

Pour cet exemple, nous allons nous intéresser à des scénarios plus complets.

La première étape consiste à réduire les contextes similaires. La figure 6.9 présente les scénarios.

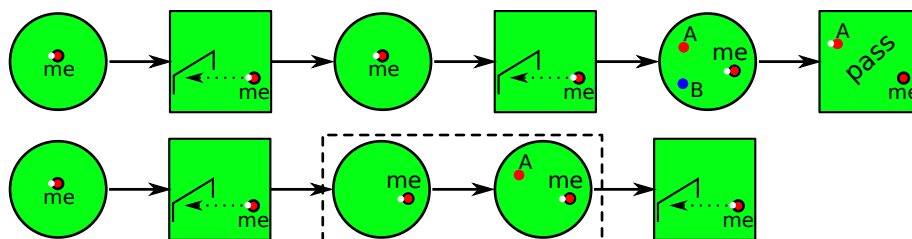


FIG. 6.9 – Regroupement des contextes

Le cadre en pointillé représente les contextes qui vont être regroupés. Nous pouvons voir en effet que le fait que l'apprenant ait la balle est inclus dans le second contexte.

La figure 6.10 représente le résultat de l'alignement plus la détection des points clés.

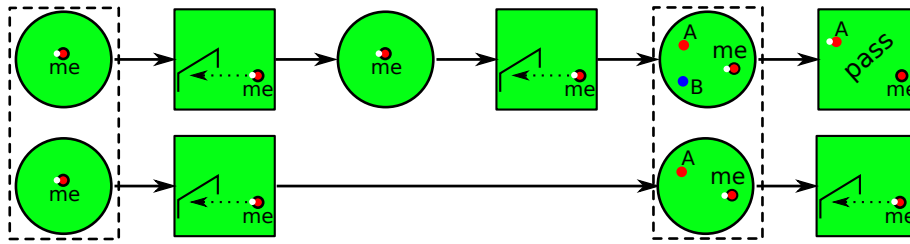


FIG. 6.10 – Regroupement des contextes

Nous pouvons voir que l'alignement a permis de sauter l'action avancer répétée dans le dernier scénario et que les points clés de la simulation ont bien été trouvés.

Il nous reste maintenant à éprouver cet algorithme sur des scénarios récupérés à partir de traces réalisées par un apprenant. Ensuite nous pourrions nous pencher sur le regroupement des scénarios en un graphe.

Chapitre 7

Conclusion

Ce travail de Master avait pour problématique l'aspect dynamique de certaines simulations développées par le CERV. Nous avons besoin d'un modèle, pour représenter les comportements non-procéduraux de l'apprenant, afin de permettre au Tuteur Intelligent (ITS) d'apporter une aide pédagogique à l'apprenant.

La modélisation du comportement d'un utilisateur se servant d'une application est le but des recherches menées dans le domaine de la Programmation Par Démonstration (PBD). Le chapitre 2 a présenté comment les systèmes de PBD modélisent le comportement d'un utilisateur et comment ils peuvent apprendre et ré-utiliser ces comportements.

Si les systèmes de PBD permettent de modéliser des situations linéaires, les situations dynamiques sont caractérisées par le fait que l'apprenant peut avoir à changer de stratégie à n'importe quel moment pour s'adapter à un changement de situation. Les psychologues, nous disent que ces changements de situations sont identifiables par la notion de contexte.

Nos *Dynamic Contextual Graphs* (DCxG), décrits dans le chapitre 3, intègrent cette notion de contexte pour permettre la modélisation de stratégies dans un environnement dynamique.

Le chapitre 4, consacré à l'utilisation des DCxG, nous a permis de voir comment exploiter les informations fournies pour pouvoir prendre des décisions dans un environnement dynamique. Nous avons aussi proposé une approche permettant d'apporter une assistance à un apprenant utilisant la simulation.

Dans le dernier chapitre, nous avons pu voir que, si notre modèle apportait des solutions quant à la prise de décision dans un environnement dynamique, il restait tout de même des problèmes dans le cadre de la collaboration.

Le travail mené dans le cadre de la simulation de football m'a conduit à participer à la rédaction d'un article avec R. BENARD et P. DE LOOR. Cet article a été accepté pour les journées de l'Aber Wrac'h, conférence organisée par le LISYC et proposé pour la conférence IRI'06 traitant de la réutilisation de l'information.

Ce stage de master a aussi été l'occasion d'apprécier la vie dans un laboratoire de recherche, de rencontrer des chercheurs, de pouvoir partager et défendre mes idées.

Dans le cadre d'un travail ultérieur, il serait intéressant de pouvoir finir l'implémentation de l'algorithme d'apprentissage et de pouvoir le tester sur des scénarios réalisés par un formateur ou un apprenant.

Il restera aussi à traiter la collaboration d'un point de vue conceptuel, ce qui impliquera certainement par la suite le développement d'un système de communication entre les joueurs dans la simulation de football.

Bibliographie

- [1] Mathias Bauer. Acquisition of user preferences for plan recognition. pages 105–112, 1996.
- [2] R. Benard, M. Aubry, and P. De Loor. Le contexte : un vecteur d'information pour les situations dynamiques et collaboratives. In *Journées de l'Aber Wrac'h*, 2006.
- [3] R. Benard, P. De Loor, and J. Tisseau. Understanding dynamic situations through context explanation. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies, ICALT*, 2006.
- [4] Y. Bengio and P. Frasconi. Input-output HMM's for sequence processing. *IEEE Transactions on Neural Networks*, 7(5) :1231–1249, September 1996.
- [5] Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press, 1995.
- [6] Steven Bogaerts and David B. Leake. Facilitating cbr for incompletely-described cases : Distance metrics for partial problem descriptions. In *ECCBR*, pages 62–76, 2004.
- [7] P. Brézillon. Context dynamic and explanation in contextual graphs. In Patrick Blackburn, Chiara Ghidini, Roy M. Turner, and Fausto Giunchiglia, editors, *Modeling and Using Context : Fourth International and Interdisciplinary Conference, Context 2003*, pages 94–106, Berlin, 2003. Springer-Verlag.
- [8] Allen Cypher. Eager : Programming repetitive tasks by example. In *Proceedings of CHI'91*, pages 33–39, 1991.
- [9] Allen Cypher. *Watch What I Do. Programming by Demonstration*. MIT Press, 1993. 652 pages.
- [10] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a cappella : programming by demonstration of context-aware applications. In *CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40, New York, NY, USA, 2004. ACM Press.
- [11] Michael Fagan and Pádraig Cunningham. Case-based plan recognition in computer games, 2003.
- [12] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4) :384–421, 2000.
- [13] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog : learning procedures for technical support. In *IUI '04 : Proceedings of the 9th international conference on Intelligent user interface*, pages 109–116, New York, NY, USA, 2004. ACM Press.
- [14] Tessa Lau, Steven A. Wolfman, Pedro Domingos, and Daniel S. Weld. Programming by demonstration using version space algebra. *Mach. Learn.*, 53(1-2) :111–156, 2003.
- [15] Tessa A. Lau and Daniel S. Weld. Programming by demonstration : An inductive learning formulation. In *Intelligent User Interfaces*, pages 145–152, 1999.
- [16] David Maulsby. Inductive task modeling for user interface customization. In *IUI '97 : Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 233–236, New York, NY, USA, 1997. ACM Press.
- [17] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [18] Dicky Suryadi and Piotr J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *UM '99 : Proceedings of the seventh international conference on User modeling*, pages 223–232, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.
- [19] Annika Wærn and Ola Stenborg. A simplistic approach to keyhole plan recognition. 1995.