LA RECONNAISSANCE DE PLAN DES ADVERSAIRES

par

Francis Bisson

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 19 juillet 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-91043-6

Our file Notre référence
ISBN: 978-0-494-91043-6

Le 6 août 2012

le jury a accepté le mémoire de Monsieur Francis Bisson
dans sa version finale.

Membres du jury

Professeur Froduald Kabansa
Directeur de recherche
Département d'informatique

Monsieur Abder Rezak Benaskeur
Codirecteur de recherche
Recherche et développement pour la Défense Canada - Valcartier

Monsieur Hengameh Irandoust
Codirecteur de recherche
Recherche et développement pour la Défense Canada - Valcartier

Professeur Sylvain Giroux
Membre
Département d'informatique

Professeur Hugo Larochelle
Président rapporteur
Département d'informatique

Sommaire

Ce mémoire propose une approche pour la reconnaissance de plan qui a été conçue pour les environnements avec des adversaires, c'est-à-dire des agents qui veulent empêcher que leurs plans soient reconnus. Bien qu'il existe d'autres algorithmes de reconnaissance de plan dans la littérature, peu sont adaptés pour de tels environnements. L'algorithme que nous avons conçu et implémenté (PROBE, *Provocation for the Recognition of Opponent BEhaviours*) est aussi capable de choisir comment provoquer l'adversaire, en espérant que la réaction de ce dernier à la provocation permette de donner des indices quant à sa véritable intention. De plus, PROBE utilise des machines à états finis comme représentation des plans, un formalisme différent de celui utilisé par les autres approches et qui est selon nous mieux adapté pour nos domaines d'intérêt. Les résultats obtenus suite à différentes expérimentations indiquent que notre algorithme réussit généralement à obtenir une bonne estimation des intentions de l'adversaire dès le départ et que cette estimation s'améliore lorsque de nouvelles actions sont observées. Une comparaison avec un autre algorithme de reconnaissance de plan démontre aussi que PROBE est plus efficace en temps de calcul et en utilisation de la mémoire, sans pourtant sacrifier la qualité de la reconnaissance. Enfin, les résultats montrent que notre algorithme de provocation permet de réduire l'ambiguïté sur les intentions de l'adversaire et ainsi améliorer la justesse du processus de reconnaissance de plan en sélectionnant une provocation qui force l'adversaire, d'une certaine façon, à révéler son intention.

Mots-clés: Intelligence artificielle; raisonnement probabiliste; reconnaissance de plan; environnement avec des adversaires; jeux de stratégie en temps réel.

Remerciements

Je voudrais tout d'abord remercier Froduald Kabanza ainsi que Abder Rezak Benaskeur et Hengameh Irandoust d'avoir accepté de diriger mes travaux de recherche. Sans leur grande disponibilité, leurs commentaires toujours judicieux et la confiance qu'ils m'ont accordée, l'accomplissement de ce projet n'aurait pas été possible.

J'aimerais aussi remercier tous mes collègues étudiants et professionnels de recherche du laboratoire PLANIART pour leur aide durant plusieurs tâches de ce projet, mais aussi pour leur soutien moral. Sans eux, les longues soirées et fins de semaine passées à travailler au laboratoire auraient été beaucoup plus monotones.

J'aimerais également remercier mes parents, François et Nicole, ainsi que mon frère, Mathieu, de m'avoir toujours encouragé à poursuivre mes études et à entreprendre de nouveaux projets, et ce, malgré que j'aie souvent négligé de consacrer du temps à ma famille ces dernières années.

Enfin, je voudrais souligner l'aide financière qui m'a été accordée par le Fonds québécois de la recherche sur la nature et les technologies (FQRNT), le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) ainsi que Fujitsu Conseil (Canada), Inc.

Abréviations

APM Actions par minute

C2 Commandement et contrôle (*Command and Control*)

CSP Problème de satisfaction de contraintes (*Constraint Satisfaction Problem*)

DEC-POMDP Processus de décision de Markov partiellement observable décentralisé (*Decentralized Partially Observable Markov Decision Process*)

HMM Modèle de Markov caché (*Hidden Markov Model*)

HTN Réseau de tâches hiérarchiques (*Hierarchical Task Network*)

I-POMDP Processus de décision de Markov partiellement observable interactif (*Interactive Partially Observable Markov Decision Process*)

MDP Processus de décision de Markov (*Markov Decision Process*)

PHATT *Probabilistic Hostile Agent Task Tracker*

POMDP Processus de décision de Markov partiellement observable (*Partially Observable Markov Decision Process*)

PROBE *Provocation for the Recognition of Opponent BEhaviours*

RTS Stratégie en temps réel (*Real-Time Strategy*)

SVM Séparateur à vaste marge (*Support Vector Machine*)

Yappr *Yet Another Probabilistic Plan Recognizer*

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

Mise en contexte

L'intelligence artificielle est une discipline de l'informatique s'intéressant à la conception d'entités intelligentes [67]. Cependant, les opinions divergent quant à la définition de la notion d'intelligence d'un système automatisé. En effet, certains parlent de systèmes devant agir ou penser comme des humains (comportement vérifiable par le test de Turing [82], par exemple), alors que d'autres parlent plutôt de systèmes devant agir ou penser rationnellement, c'est-à-dire obtenir les meilleurs résultats ou arriver aux meilleures conclusions avec l'information disponible.

Dans tous les cas, il a été démontré que la planification, soit le processus délibératif visant à choisir et à organiser des actions en une structure de plan en vue d'atteindre un objectif [39], réside au cœur du raisonnement cognitif humain [76, 84]. En effet, les humains, dès un très jeune âge, planifient leurs actions à l'avance et anticipent les résultats de leurs actions, consciemment ou non, afin de résoudre des problèmes et d'accomplir des tâches plus ou moins complexes. De plus, Schmidt [72] a établi que les humains infèrent et formulent des hypothèses sur les plans et les buts des personnes avec qui ils interagissent, et raisonnent ensuite à partir de ces hypothèses.

La compréhension des plans des autres personnes s'avère donc être primordiale dans les processus cognitifs humains [10], mais aussi dans la conception de systèmes intelligents [71]. Ce problème s'appelle la *reconnaissance de plan* et peut être vu, sur le plan conceptuel, comme étant le problème inverse de la planification : en planification, le raisonnement se fait à partir d'un but pour trouver les actions (le plan) permettant de l'atteindre, alors qu'en reconnaissance de plan, le raisonnement se fait à partir de l'observation d'actions pour inférer le plan ainsi que l'objectif poursuivi.

Description du problème

Dans cette section, nous définissons tout d'abord le problème de reconnaissance de plan, son fonctionnement général et l'utilité d'un tel algorithme. Ensuite, une classification des différents types de reconnaissance de plan est donnée.

La reconnaissance de plan est le processus de compréhension des plans d'agents observés : il s'agit donc essentiellement d'un problème d'abduction logique [25] puisqu'il faut inférer les causes (les plans) à partir des effets (les actions observées). La figure 1 montre un exemple de configuration d'un problème de reconnaissance de plan, avec un agent (humain, robot, système intelligent, etc.) qui utilise l'information obtenue de ses capteurs pour exécuter des actions dans l'environnement, ainsi qu'un observateur qui examine ces actions afin de reconnaître les intentions de l'agent qu'il observe. Les plans réalisés par l'agent observé (et reconnus par l'observateur) caractérisent autant les objectifs qu'il poursuit ou l'intention qu'il souhaite réaliser (le *quoi*) que les actions qu'il doit exécuter dans l'environnement pour y parvenir (le *comment*). Selon les algorithmes et les domaines d'application, les plans peuvent être décrits selon différents formalismes, par exemple les réseaux de tâches hiérarchiques (*Hierarchical Task Networks*, ou HTN), les machines à états finis, les grammaires probabilistes, etc.

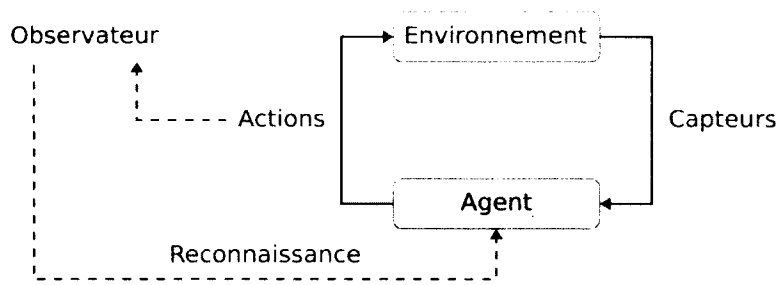


Figure 1 – Exemple de configuration d'un problème de reconnaissance de plan

Certaines approches nécessitent *a priori* la connaissance de l'ensemble des plans que l'agent observé peut exécuter et que l'observateur peut reconnaître (approches non génératives), alors que d'autres approches n'ont besoin que d'une description des actions primitives et de l'ensemble des buts possibles de l'agent observé (approches génératives). Certaines méthodes ont aussi besoin de la séquence complète des ob-

servations des actions de l'agent pour reconnaître le plan en différé (*offline*), alors que d'autres peuvent le faire en direct (*online*), en mettant à jour de manière incrémentale le modèle de reconnaissance lorsque de nouvelles observations sont obtenues. Nous reviendrons sur ces caractéristiques dans la revue de la littérature.

Dans le cas le plus simple, il n'y a qu'un seul agent exécutant des plans et un seul observateur (figure 2(a)), mais nous pouvons généraliser le problème au contexte multiagent. Ainsi, il est tout à fait possible qu'un observateur doive comprendre les intentions de plusieurs agents (figure 2(b)), que plusieurs observateurs tentent de synchroniser leur compréhension d'un seul agent (figure 2(c)), ou encore, dans le cas le plus général, que plusieurs observateurs veulent reconnaître les plans de plusieurs agents (figure 2(d)). Ces agents peuvent être coopératifs, neutres ou même hostiles envers les observateurs : plus de détails sur cette classification seront donnés dans un instant. Nous pourrions même imaginer une combinaison de ces types d'agents, par exemple un système automatique devant à la fois reconnaître les intentions d'adversaires hostiles et celles d'alliés coopératifs. Dans le reste de ce mémoire, nous nous intéressons au problème de reconnaissance d'un seul agent par un seul observateur.

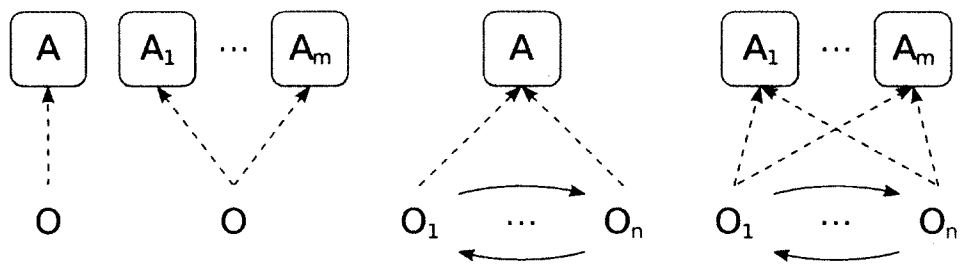


Figure 2 – Reconnaissance de plan multiagent

Étant donné que l'objectif de la reconnaissance de plan est d'arriver à comprendre les intentions d'agents et que cette information est inconnue, les observateurs doivent utiliser les observations des actions que les agents exécutent dans l'environnement. Ces observations constituent la principale entrée d'un algorithme de reconnaissance de plan. Selon les domaines d'application et les types d'agents auxquels les observateurs sont confrontés, la séquence d'observations peut être incomplète (actions importantes manquantes) ou imparfaite (observations erronées ou bruitées).

Types de reconnaissance de plan

Il existe trois catégories de domaines d'application pour la reconnaissance de plan, selon la relation existant entre l'observateur et l'agent observé. Cohen et al. [26] ont d'abord identifié les deux premiers types de problèmes de reconnaissance de plan.

En premier lieu, la reconnaissance de plan intentionnelle (*intended*) concerne un observateur et un agent observé évoluant dans le même environnement. Ils ont donc la possibilité de communiquer (directement ou indirectement), les actions de l'un peuvent influencer celles de l'autre, etc. Par exemple, l'observateur peut demander à l'agent d'exécuter certaines actions afin d'expliciter son intention, et ainsi permettre à l'observateur de raffiner sa compréhension des intentions de l'agent. L'agent observé fait un effort supplémentaire pour accomplir ses tâches avec une intention claire, précise et transparente en tête afin de ne pas faire d'erreur en suivant son plan ni d'exécuter d'actions superflues. De plus, l'agent observé est coopératif avec son observateur : il exécute ses actions de façon à ce qu'elles puissent facilement être observées, interprétées et comprises par l'observateur. Ces caractéristiques font en sorte qu'il est plus facile de résoudre les problèmes de reconnaissance de plan de ce type. La reconnaissance de plan intentionnelle est typiquement utilisée dans des contextes de collaboration humain-machine [53] ou de compréhension du langage naturel [22].

Deuxièmement, la reconnaissance externe (*keyhole*[1]) concerne plutôt un observateur et un agent observé demeurant dans des environnements séparés. Ainsi, l'observateur est confiné à surveiller l'agent sans pouvoir interagir avec lui de quelque façon que ce soit. L'agent ignore généralement qu'il est observé, éliminant du coup le besoin d'expliciter la réalisation de ses intentions. Par conséquent, les actions qu'il exécute sont normalement claires pour lui (trivialement), mais pas nécessairement pour l'observateur. Selon les domaines, l'agent peut aussi faire des erreurs en tentant de réaliser son plan. L'hypothèse de rationalité ne tient plus nécessairement : il n'est plus possible de supposer que toutes les actions observées aient un lien direct avec l'intention de l'agent. Tous ces facteurs rendent le problème de reconnaissance de plan externe beaucoup plus difficile que le problème de reconnaissance intentionnelle.

1. L'adjectif *keyhole* utilisé dans la littérature anglophone provient de l'analogie avec une personne (l'observateur) en espionnant une autre (l'agent observé) à travers le trou de la serrure d'une porte, sans avoir la possibilité d'influencer l'environnement de ce dernier.

mais aussi plus intéressant, ce qui explique le grand nombre de chercheurs qui s'y attaquent. Un exemple de reconnaissance de plan externe est la reconnaissance des activités de la vie quotidienne de personnes souffrant de la maladie d'Alzheimer dans des habitats intelligents [20, 66] : le patient (l'agent observé) peut exécuter ses plans de façon incohérente ou erronée et l'observateur n'interagit pas directement avec le patient afin d'encourager son autonomie.

Finalement, Carberry [22] et, plus tard, Geib et Goldman [34] ont distingué un troisième cas où l'agent observé est hostile au processus de reconnaissance de ses intentions. Il s'agit de la reconnaissance de plan des adversaires (*adversarial*) ou tactique [55], et c'est bien entendu le cas qui nous intéresse ici. Un adversaire nuit délibérément à la reconnaissance de ses intentions en utilisant la déception [1] (comportement trompeur) et la confusion, et en niant l'accès à l'information (*information denial*). En effet, ce n'est pas dans son intérêt que ses intentions soient reconnues par l'observateur, son rival. Un adversaire tente aussi d'empêcher l'observateur (et ses alliés, dans un contexte multiagent) d'accomplir ses objectifs, diamétralement opposés aux siens. Ce type de reconnaissance de plan se trouve par exemple dans des applications militaires [4, 7] ou dans des jeux vidéo où l'objectif est de battre les adversaires [45, 51].

Le tableau 1 résume la classification des différents types de reconnaissance de plan, selon la relation existant entre l'observateur et l'agent observé et l'attitude de ce dernier envers son observateur.

Tableau 1 – Types de reconnaissance de plan

Environnement(s)	Attitude	Type de reconnaissance
Distincts	—	Externe (*keyhole*)
Même	Coopérative	Intentionnelle (*intended*)
Même	Hostile	Des adversaires (*adversarial*)

Provoquer l'adversaire pour observer sa réaction

Les algorithmes de reconnaissance de plan en direct sont typiquement *passifs*, laissant toute l'initiative à l'agent observé : ils attendent que ce dernier agisse avant de

pouvoir mettre à jour le modèle de reconnaissance en tenant compte des nouvelles observations afin de raffiner leur compréhension des plans de l'agent observé [2]. Cela peut être fatal dans un domaine hautement dynamique avec des adversaires où l'observateur doit prendre des décisions en temps réel en fonction des plans de l'adversaire : ce problème est exacerbé par un environnement partiellement observable.

Une façon de pallier ce problème consiste à exécuter des actions en espérant que la réaction de l'adversaire à ces actions fournisse à l'observateur des indices quant à ses intentions réelles. Ainsi, l'observateur prend l'initiative dans le processus de reconnaissance de plan, ce qui pourrait lui permettre de comprendre plus rapidement les intentions de l'adversaire. Notons cependant que l'idée d'éliciter une réaction chez l'agent observé pourrait aussi s'appliquer dans le cadre de la reconnaissance de plan intentionnelle, où deux agents collaboratifs veulent communiquer sans pouvoir le faire directement.

Organisation du mémoire

Nous commençons tout d'abord, dans le chapitre 1, par une description de deux domaines d'application pour notre algorithme de reconnaissance de plan des adversaires. Ensuite, nous présentons au chapitre 2 une revue de la littérature afin de mettre en contexte ce projet de recherche. Le chapitre 3 explique en détail notre algorithme de reconnaissance de plan des adversaires ainsi que notre algorithme de provocation pour observer la réaction. Ces algorithmes, programmés en entier puisqu'aucune autre implémentation n'était alors disponible, utilisent des machines à états finis comme représentation des plans, un formalisme qui est selon nous mieux adapté pour nos domaines d'intérêt. Dans le chapitre 4, nous introduisons d'abord plusieurs métriques pour mesurer et comparer la qualité des résultats produits par notre algorithme de reconnaissance de plan, puis nous analysons les résultats obtenus lors des expérimentations. Enfin, le chapitre 5 donne quelques pistes d'idées à explorer pour faire suite à ce travail de recherche avant de conclure ce mémoire.

2. Les méthodes de reconnaissance de plan en différé sont forcément passives puisqu'elles nécessitent *a priori* la séquence complète d'actions observées dans un scénario déjà terminé, à partir desquelles le plan de l'agent observé doit être inféré.

Chapitre 1

Exemples de domaines d'application

La reconnaissance de plan des adversaires peut se faire dans différents domaines d'application. Ici, nous nous intéressons plus particulièrement à deux domaines similaires mais ayant tout de même certaines différences dignes de mention. Nous décrivons donc, dans l'ordre, le domaine de l'évaluation de la menace dans la guerre navale et le domaine des jeux de stratégie en temps réel.

1.1 Évaluation de la menace dans la guerre navale

Le domaine de l'évaluation de la menace dans la guerre navale s'inscrit dans un contexte plus général de commandement et de contrôle (*Command and Control*, C2) [43, 75]. Il s'agit de « l'autorité et de la direction exercées par un commandant désigné sur des forces chargées d'accomplir une mission. Le C2 est le moyen par lequel un commandant de forces armées synchronise ou intègre les activités de ces forces pour atteindre un certain objectif. Le C2 relie toutes les fonctions et les tâches opérationnelles, et s'applique à tous les niveaux de la guerre et des échelons du commandement sur toute la gamme des opérations militaires » [83]. C'est un processus qui se divise en plusieurs étapes, comme illustré dans la figure 1.1, et que nous détaillons ici.

Un des aspects importants du processus de C2 dans la guerre navale est l'analyse de la situation tactique. Cette analyse comprend la compilation de l'image tactique,

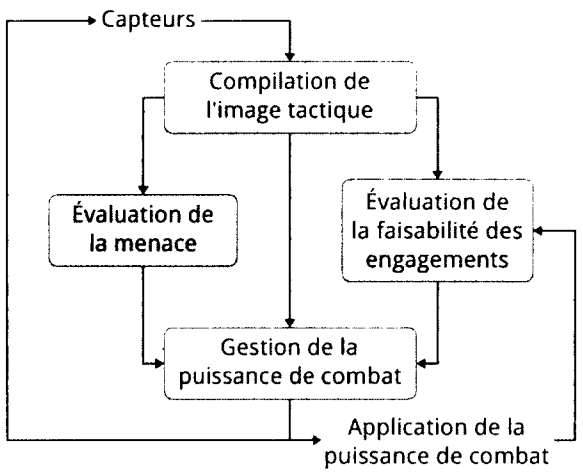


Figure 1.1 – Processus de commandement et de contrôle (C2) [43]

l'évaluation de la menace et l'évaluation de la faisabilité des engagements. La gestion de ressources de combats s'appuie sur les résultats de ces trois sous-processus pour planifier et exécuter les engagements contre les menaces [13].

La compilation de l'image tactique sert à fusionner les données brutes des différents capteurs (imparfaits et ayant parfois des angles morts) d'une plate-forme navale (ou d'un groupe opérationnel naval) afin de produire une *image tactique* identifiant les entités avec leur type, leur position, leur altitude, leur azimut, leur vélocité, l'armement et les systèmes de détection qu'ils possèdent ou qu'ils pourraient posséder, etc. La figure 1.2 illustre une image tactique simple avec une symbologie propre au domaine militaire. Les objets en bleu désignent des bateaux amis, alors que les objets en rouge désignent des entités ennemies et sont expliquées dans le tableau 1.1.

Tableau 1.1 – Symbologie militaire pour les entités ennemies

Aérienne	Surface	Sous-marine	Missile
			

À partir de ces données, d'informations connues *a priori* et de la doctrine relative aux objectifs de la mission, une seconde étape du processus de C2 consiste à évaluer le niveau de menace des entités. L'objectif de cette évaluation est de « déterminer

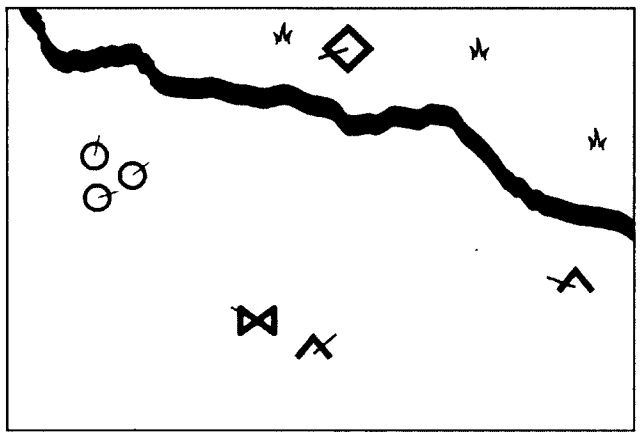


Figure 1.2 – Image tactique simplifiée

si chaque entité a le désir (intention) et possède les ressources nécessaires (capacité) pour infliger des dommages au groupe opérationnel naval ou à ses intérêts, et si l'environnement remplit les conditions nécessaires pour que le plan de l'entité puisse être complété avec succès (opportunité) » [44]. Généralement, cette évaluation résulte en une classification de chaque menace (basse, moyenne, élevée, inconnue) en se basant sur trois indicateurs : (1) l'intention, (2) la capacité et (3) l'opportunité [74].

Parallèlement à la compilation de l'image tactique et à l'évaluation de la menace, le processus d'évaluation de la faisabilité des engagements sert à analyser l'opportunité et la capacité du groupe opérationnel naval à engager le combat avec les entités jugées menaçantes. Toutes les options d'engagement doivent être considérées et ordonnées, afin de faciliter leur application si cela devient nécessaire.

À partir de l'image tactique et des résultats de l'évaluation de la menace ainsi que de l'évaluation de la faisabilité des engagements, le processus de gestion de la puissance de combat produit un plan à suivre pour éliminer les menaces et assure la bonne et correcte exécution de ce plan. La boucle de contrôle est ainsi fermée par ce processus : l'image tactique est mise à jour si de nouvelles entités sont détectées ou si des menaces sont éliminées, la menace encourue et les options d'engagement sont réévaluées, et ainsi de suite.

Il va sans dire que les commandants d'opérations navales font face à une pression immense. La charge cognitive requise pour mener ces opérations est énorme [32, 58] et

les erreurs de jugement peuvent avoir des conséquences fatales. L'automatisation de certains processus de C2 vise à réduire la charge cognitive des commandants d'opérations navales afin de les aider à accomplir leurs tâches. La conception d'un système d'aide à la décision pour l'évaluation de la menace vient aider à pallier ce problème.

Un algorithme de reconnaissance de plan permettrait d'évaluer automatiquement des indicateurs de menace. En effet, reconnaître un plan fournit implicitement à la force opérationnelle l'intention des différents agents dans l'environnement et le moyen qu'ils comptent utiliser pour réaliser leur intention. En d'autres mots, certains indicateurs de l'intention hostile d'un agent, en particulier ceux correspondants aux buts poursuivis, peuvent être calculés par un algorithme de reconnaissance de plan.

1.2 Jeux de stratégie en temps réel

Les jeux de stratégie en temps réel (*Real-Time Strategy*, RTS) sont un genre de jeux vidéo où des joueurs (humains ou artificiels) s'affrontent, l'objectif étant d'anéantir ses adversaires. Les joueurs peuvent s'affronter en duel, mais les escarmouches peuvent aussi impliquer plusieurs joueurs coalisés contre d'autres joueurs.

Les joueurs construisent des structures, recrutent des unités, récoltent des ressources, recherchent des mises à niveau technologiques et se battent pour le contrôle du territoire, tout en montant une défense pour contrer les attaques des adversaires et en préparant une armée pour les conquérir. Des ressources sont nécessaires pour construire des structures, pour recruter des unités, et ainsi de suite.

Construire des structures et rechercher des technologies permet aux joueurs de recruter des unités plus puissantes, de rechercher des mises à niveau plus sophistiquées, de construire des structures plus avancées et ainsi de suite, suivant la structure de *l'arbre technologique* (*technology tree*) [1]. Par exemple, une stratégie typique dans un jeu RTS ayant une thématique de science-fiction pourrait être de se concentrer sur le recrutement soit d'unités souterraines, soit d'unités aériennes : les joueurs suivant une de ces stratégies doivent acquérir les ressources et rechercher les mises à niveau

1. Les arbres technologiques sont des structures sous forme de graphes acycliques dirigés (souvent de simples arbres) identifiant hiérarchiquement les prérequis pour les unités, les structures et les mises à niveau technologiques.

technologiques requises par les structures de recrutement, suivant le bon chemin dans l'arbre technologique (figure 1.3).

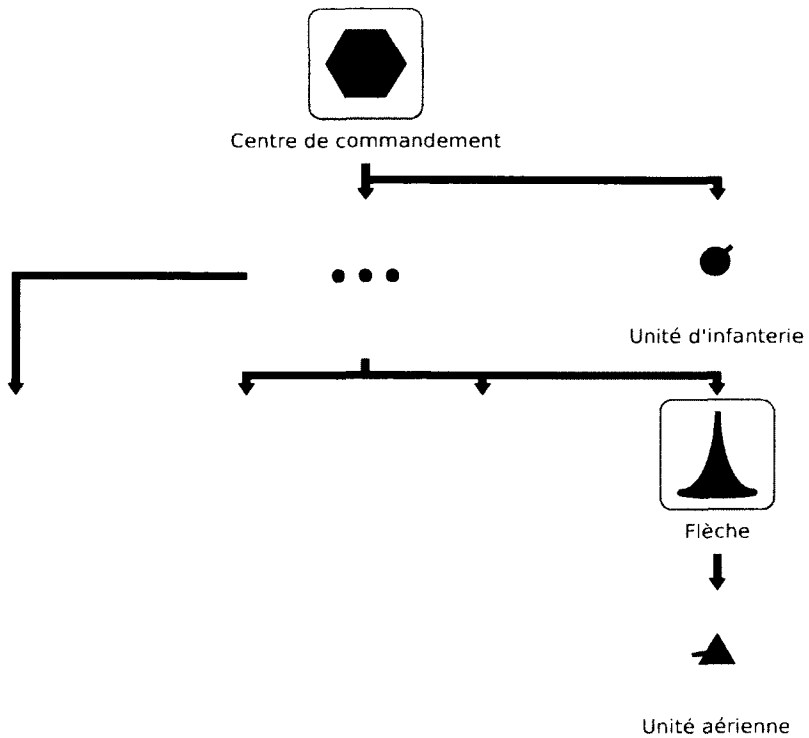


Figure 1.3 – Extrait du chemin dans un arbre technologique pour une stratégie axée sur le recrutement d'unités aériennes

Durant une partie, les joueurs doivent soigneusement équilibrer leur économie afin de maximiser la production d'unités. Dans le vocabulaire RTS, cela s'appelle la *macrogestion* et comprend aussi bien décider où et quand construire une base secondaire où les ressources abondent afin d'augmenter les revenus, la production d'unités et ainsi de suite. La *microgestion* concerne plutôt la capacité à contrôler individuellement les unités et à remplacer leurs comportements par défaut, sous-optimaux et imparfaits. Par exemple, la figure 1.4 montre un joueur contrôlant une unité blessée ($\mathbf{U1}_A$) pour qu'elle se replie derrière la ligne de défense en attendant d'être soignée afin d'éviter de se faire abattre par les ennemis en rouge.

Afin de gagner une partie, il est crucial pour un joueur d'être capable de reconnaître les comportements de macrogestion et de microgestion des adversaires. En

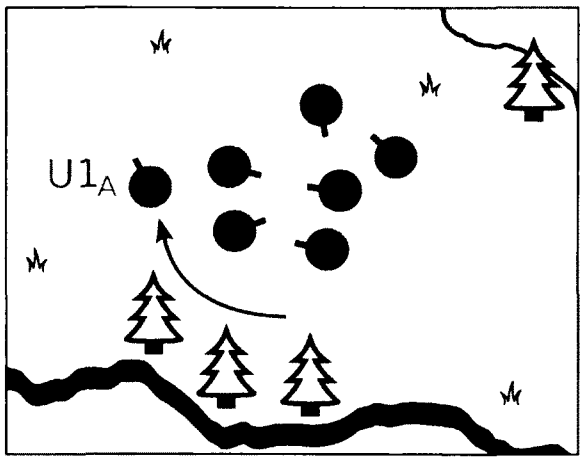


Figure 1.4 – Microgestion d'une unité blessée

effet, savoir où et quand un adversaire prévoit construire une base secondaire permet à un joueur de capturer la position avant que l'adversaire ne puisse le faire, ou encore d'attaquer la base principale de l'adversaire pendant qu'elle est temporairement vulnérable[2]. De plus, être capable de reconnaître de quelle façon les groupes d'unités ennemies vont diriger leurs attaques permet à un joueur de gagner un avantage tactique sur ses adversaires. Dans les deux cas, ce sont des problèmes de reconnaissance de plan des adversaires.

La zone de visibilité de la carte du jeu dépend des unités et des structures d'un joueur. En effet, chaque entité a son propre rayon de visibilité, selon son type et les mises à niveau technologiques que le joueur possède. Aussi, certaines unités ont des habiletés spéciales d'invisibilité (temporaire ou permanente), et ne peuvent être détectées et attaquées que si l'adversaire possède la technologie requise (par exemple, des structures spéciales de détection).

Le temps est bien entendu un aspect très important des jeux RTS. Les plans des joueurs changent continuellement, des actions simultanées sont sans cesse exécutées[3].

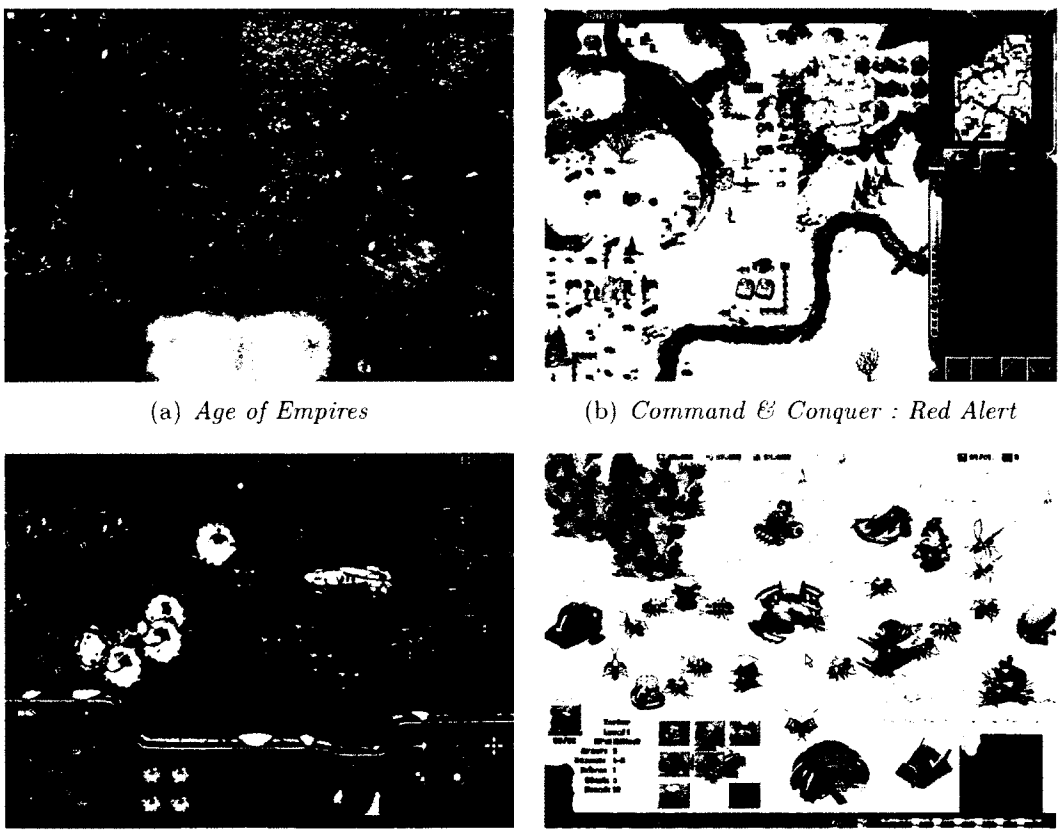
2. Les unités responsables de construire une base secondaire sont habituellement escortées par des unités défensives et offensives quittant la base principale pour défendre la nouvelle base qui s'apprête à être construite.

3. Les joueurs professionnels de *StarCraft : Brood War*, par exemple, exécutent en moyenne entre 200 et 250 actions par minute (APM) ; les joueurs contrôlés par un ordinateur ne sont virtuellement limités que par la fréquence d'horloge du microprocesseur.

et la situation tactique peut évoluer très rapidement. Un joueur ayant un avantage initialement significatif sur ses adversaires peut vite devenir vulnérable suite à une bévue, aussi anodine puisse-t-elle sembler.

Le rythme élevé des jeux RTS combiné aux contraintes de visibilité oblige les joueurs à continuellement faire de l'observation active afin de maintenir une compréhension suffisante de la situation. Un joueur ayant exploré une région il y a un certain temps à l'aide d'un éclaireur qui a ensuite quitté peut s'attendre à ce que la situation tactique ait évolué, peut-être même de façon très drastique. Un joueur doit donc régulièrement envoyer des éclaireurs pour explorer les territoires contrôlés par les adversaires afin de maintenir sa compréhension de la situation à jour.

Des exemples de jeux RTS populaires sont illustrés dans la figure 1.5.



(a) *Age of Empires* (b) *Command & Conquer : Red Alert*

(c) *StarCraft : Brood War* (d) *Magnant* (conçu avec *Stratagus* [60])

Figure 1.5 – Exemples de jeux RTS populaires

1.3 Discussion

Les domaines de l'évaluation de la menace dans la guerre navale et des jeux de stratégie en temps réel se ressemblent sur plusieurs points, en particulier sur le fait qu'ils concernent tous les deux la mise en œuvre de plans d'action pour contrer des menaces dans un environnement incertain. Ils sont suffisamment similaires pour permettre un transfert des algorithmes que nous développons d'un domaine à l'autre.

Pour ce mémoire, nous avons opté pour l'expérimentation de nos algorithmes avec les jeux de stratégie en temps réel, en particulier le jeu *StarCraft : Brood War*. Ce domaine offre l'avantage évident d'être beaucoup plus facile à utiliser pour les expérimentations et pour lequel concevoir des scénarios. D'ailleurs, plusieurs scénarios de reconnaissance de plan dans les jeux RTS sont détaillés dans l'annexe A. Les jeux RTS sont aussi une plate-forme très intéressante pour une multitude d'autres problèmes de recherche en intelligence artificielle [21], favorisant ainsi la synergie avec d'autres projets liés aux mécanismes d'intelligence artificielle.

De façon générale, le problème de prise de décision dans un environnement simultané, multijoueur, à somme nulle et avec de l'information imparfaite (comme les jeux RTS) est abordé par la théorie des jeux [47, 56]. Cependant, étant donné l'aspect temps réel et simultané du jeu et le grand nombre d'actions possibles pour chaque unité à contrôler, le problème est beaucoup trop complexe pour être résolu parfaitement. Des techniques d'échantillonnage aléatoire ou de simulation [49, 69] sont souvent utilisées pour trouver des stratégies acceptables, mais non optimales.

Afin de pallier cet important problème, le laboratoire PLANIART étudie présentement une approche novatrice [23] utilisant des heuristiques de planification régressive basées sur un graphe de planification (*planning graph*) [19]. Cet algorithme prend en entrée la stratégie de l'adversaire, identifiée par un algorithme de reconnaissance de plan, afin de guider la recherche vers une stratégie susceptible de contrer efficacement celle de l'ennemi, c'est-à-dire un plan contenant des actions qui invalident les futures actions prédites de l'adversaire.

Chapitre 2

État de l'art

Les algorithmes de reconnaissance de plan présents dans la littérature sont traditionnellement classifiés en deux catégories : les approches symboliques (logiques) et les approches probabilistes. Les deux écoles de pensée ont cependant plusieurs limitations pour les applications de la reconnaissance de plan. Dans ce chapitre, nous discutons quelques approches de reconnaissance de plan, en particulier en ce qui concerne leur applicabilité pour notre domaine d'intérêt. Nous présentons d'abord les approches symboliques puis les approches probabilistes. Ensuite, nous discutons des méthodes hybrides visant à combiner les avantages des deux familles d'approches. Nous présentons par la suite les approches génératives de reconnaissance de plan, un nouvel axe de recherche s'appuyant sur des principes de planification automatique en intelligence artificielle pour simuler le processus mental de l'agent observé. Enfin, nous examinons la possibilité de provoquer l'agent observé afin d'aider le processus de reconnaissance de plan à comprendre les intentions de ce dernier.

Notons que bien que les jeux de stratégie en temps réel soient un domaine multiagent, nous nous intéressons au problème de reconnaissance d'un seul agent contrôlant plusieurs unités simultanément. Notre recherche ne s'intéresse donc pas aux interactions entre les unités collaborant pour atteindre des buts communs, ni à reconnaître la composition (statique ou dynamique) des groupes ou des équipes d'unités, deux sous-problèmes de la reconnaissance de plan multiagent [6, 12, 68, 78, 80]. Les approches proposées pour ce problème ne seront donc pas discutées dans ce chapitre.

2.1 Approches symboliques

Les travaux de Kautz et Allen [46] sont à la base de bien des recherches en reconnaissance de plan. Les auteurs ont initialement abordé le problème de reconnaissance de plan comme un problème d'identification d'un ensemble minimal d'actions de haut niveau dans une taxinomie d'actions et de sous-actions prédéfinie (voir la figure 2.1) pour expliquer un ensemble d'actions observées. Le problème revient donc à calculer

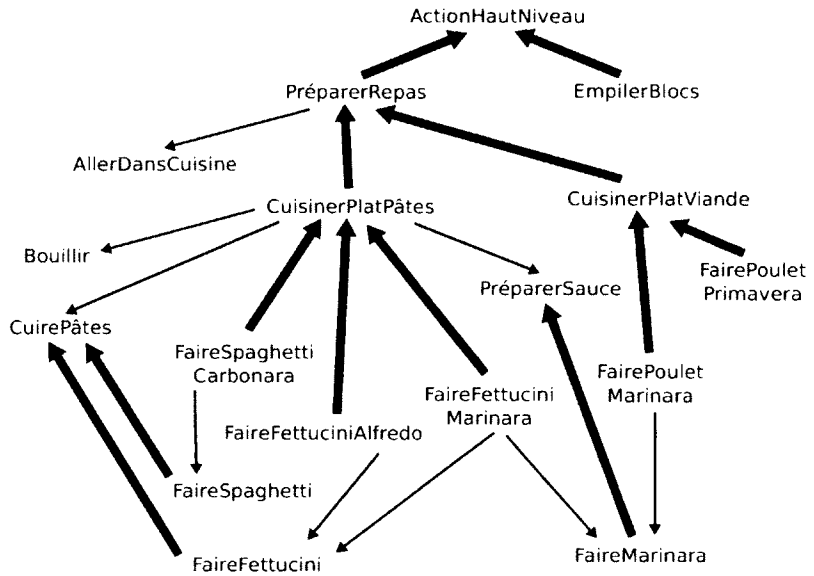


Figure 2.1 – Taxinomie d'actions de Kautz et Allen [46]

la couverture minimale du graphe de la bibliothèque de plans (la taxinomie d'actions) correspondant aux observations, formalisé par la circonscription de McCarthy [54]. Informellement, cela revient à appliquer le principe du rasoir d'Occam et considérer que les hypothèses explicatives les plus simples sont aussi les plus vraisemblables.

Cela n'est pourtant pas toujours adéquat, en particulier lors d'un diagnostic médical (des symptômes s'expliquent peut-être mieux par deux maladies fréquentes que par une seule maladie plus rare [24]), lorsque les agents observés ne sont pas rationnels (par exemple dans le domaine de la reconnaissance d'activités de personnes atteintes de déficiences cognitives [66]), ou encore lorsqu'ils ont des comportements trompeurs. En effet, comme l'ont indiqué Elsaesser et Stech [29], la déception fonc-

tionne parce qu'elle exploite les erreurs de raisonnement, les limitations cognitives et les biais concomitants : cela inclut tirer des conclusions hâtives et ignorer des hypothèses de déception. La lacune majeure des méthodes de reconnaissance de plan purement symboliques est ainsi dévoilée : elles sont incapables de préférer une hypothèse à une autre qui expliquerait moins bien les observations (à complexité égale, dans le cas du système de Kautz et Allen).

Plus récemment, Avrahami-Zilberbrand et Kaminka [5] ont proposé un algorithme de reconnaissance de plan purement symbolique et complet, c'est-à-dire qu'il génère toutes les hypothèses cohérentes avec la séquence d'observations. Les auteurs ont donc choisi d'ignorer entièrement le problème de classement des hypothèses ; ils soutiennent cependant qu'il serait possible d'apposer un modèle probabiliste à leur algorithme afin de pallier ce problème, à la manière des méthodes hybrides (section 2.3). L'algorithme suppose la présence d'une bibliothèque de plans similaire à celle de Kautz et Allen détaillant de façon hiérarchique les plans possibles et les actions les composant, mais est aussi capable de traiter les observations formées à partir de plusieurs caractéristiques, pas seulement celles correspondant à des activités plus ou moins abstraites. La principale contribution de cette méthode est son efficacité algorithmique. En effet, l'algorithme calcule implicitement les hypothèses explicatives en étiquetant avec des estampilles temporelles les nœuds de la bibliothèque de plans correspondant aux observations, évitant ainsi de générer explicitement les hypothèses dans des structures externes, processus coûteux en temps et en mémoire lorsque le nombre d'hypothèses est grand. L'algorithme est capable de générer explicitement les hypothèses sur demande, lorsqu'elles sont requises. Bien que l'efficacité algorithmique de cette méthode soit une caractéristique attrayante, son apport réel reste à confirmer dans un domaine hautement dynamique comme les jeux de stratégie en temps réel, où les hypothèses sont constamment requises par les processus de prise de décision afin que l'agent puisse rapidement et convenablement réagir à la situation présente.

2.2 Approches probabilistes

En revanche, les approches purement probabilistes abordent directement le problème de classement des hypothèses explicatives grâce à la théorie des probabilités.

L'idée générale consiste à calculer la probabilité *a posteriori* des buts possibles de l'agent observé à l'aide d'un modèle stochastique, puis à raffiner cette probabilité grâce aux observations.

Charniak et Goldman [24] ont été les premiers à proposer une approche probabiliste pour la reconnaissance de plan. Leur algorithme construit dynamiquement un réseau bayésien à partir des observations et d'une base de connaissances de faits sur le monde, spécifiés en logique du premier ordre. Dans le réseau bayésien, les nœuds racine correspondent aux hypothèses sur les buts de l'agent observé : les probabilités conditionnelles de ces nœuds sont ensuite calculées (grâce au théorème de Bayes) afin de pouvoir comparer leur vraisemblance.

L'approche de Pynadath et Wellman [61] propose une architecture générale (un réseau bayésien illustré dans la figure 2.2) pour la reconnaissance de plan modélisant, entre autres, le contexte (la description de l'état du monde) et l'état mental de l'agent observé. Chaque composante de cette architecture est implémentée par un

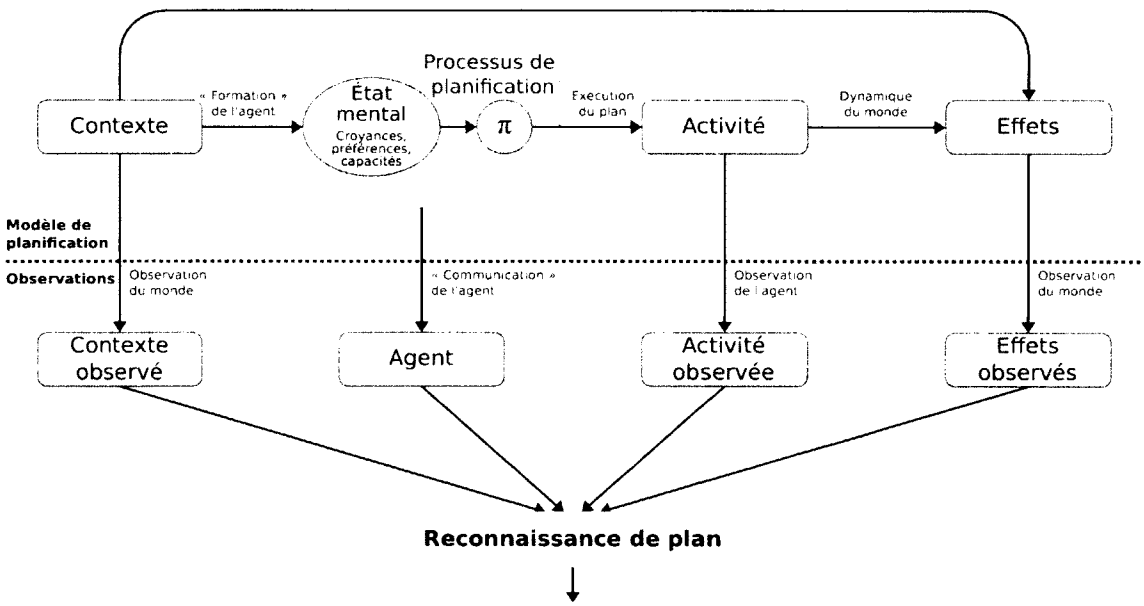


Figure 2.2 – Architecture de reconnaissance de plan de Pynadath et Wellman [61]

réseau bayésien dynamique [67], permettant ainsi de capturer leur évolution dans le temps. L'architecture est aussi capable de représenter les capteurs imparfaits ainsi que

les informations non observables en séparant ce que l'observateur perçoit du monde et le contexte réel, ou plutôt les croyances de l'agent observé sur la valeur réelle des variables décrivant le contexte. En effet, comme l'ont noté Pynadath et Wellman, un agent rationnel prendra ses décisions en fonction de sa propre perception (possiblement erronée) du monde : ce sont donc ces croyances que l'observateur doit modéliser pour parvenir à simuler correctement l'état mental de l'agent observé. Notons que la modélisation de l'état mental se limite à un seul niveau, et n'est pas infiniment récursive comme dans la théorie des jeux [56]. Néanmoins, cette méthode jette les bases pour les approches génératives de reconnaissance de plan, que nous décrirons dans la section 2.4.

Bien qu'ils permettent de régler le problème de classement des hypothèses, les approches de ce type nécessitent une grande quantité de valeurs devant être connues à l'avance, en particulier les probabilités *a priori* des variables aléatoires ainsi que les tables de probabilités conditionnelles entre les variables dépendantes. Ces valeurs sont généralement difficiles à obtenir et la précision des probabilités *a posteriori* est fortement liée à l'exactitude de ces paramètres. Certaines valeurs peuvent être estimées grâce à des méthodes d'apprentissage automatique, mais une grande quantité de données d'apprentissage devient alors nécessaire [2]. Les calculs requis pour mettre à jour les réseaux bayésiens sont aussi généralement très coûteux [24]. Ce problème est exacerbé lorsque de nouveaux nœuds sont ajoutés dynamiquement aux réseaux afin de tenir compte des nouvelles observations : les calculs inhérents à la complexité des réseaux rendent difficilement utilisables ces approches à long terme dans un domaine comme les jeux de stratégie en temps réel.

2.3 Approches hybrides

En raison des limitations évoquées plus tôt, la majorité des travaux récents en reconnaissance de plan portent plutôt sur des approches hybrides, combinant les avantages des méthodes symboliques et probabilistes. Sukthankar et Sycara [77] ont proposé une approche différente des méthodes probabilistes pour résoudre le problème de classement des hypothèses explicatives. Leur algorithme utilise d'abord un séparateur à vaste marge (SVM) et un modèle de Markov caché (*Hidden Markov Model*, HMM)

pour classifier des données brutes de capture de mouvements en activités prédéterminées[1]. L'algorithme compare enfin les traces d'activités observées ainsi générées aux plans dans la bibliothèque, modélisés sous forme de graphes dirigés (figure 2.3), afin de reconnaître ceux qui sont présentement exécutés par l'agent observé.

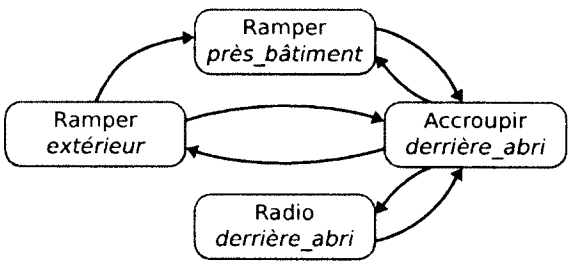


Figure 2.3 – Plan de surveillance [77]

Afin d'identifier l'hypothèse la plus plausible, une fonction de coût est établie sur les transitions entre les comportements (lorsque l'agent observé abandonne un comportement pour commencer à en exécuter un autre), et la meilleure hypothèse est celle qui minimise la somme des coût sur la séquence d'observations. Une fonction de coût naïve consiste à donner un coût nul lorsque l'agent continue d'exécuter le même comportement et un coût positif aux transitions entre des comportements différents : cette fonction est semblable à la minimisation du nombre de plans de haut niveau de Kautz et Allen [46].

Cependant, comme le reconnaissent les auteurs, cette fonction de coût est fortement dépendante du domaine d'application, et il peut être très difficile d'en obtenir une qui donne des résultats précis ; comme expliqué précédemment, la préférence des hypothèses les plus simples n'est pas toujours adéquate. De plus, la précision du processus de reconnaissance de plan est évidemment directement liée au choix d'une bonne fonction de coût.

Néanmoins, la flexibilité de cette fonction de coût permet d'utiliser cette approche dans différents contextes de reconnaissance de plan, comme la collaboration entre des alliés (faibles coûts vers les comportements d'entraide), la modélisation des adversaires

1. Dans le domaine utilisé par les auteurs, soient les opérations militaires en milieu urbain, ces actions sont : (1) marcher ; (2) courir ; (3) ramper ; (4) boiter ; (5) examiner ; (6) accroupir ; et (7) lever. De plus, deux autres actions sont identifiées uniquement par le bruit qu'elles produisent : (1) appeler par radio ; et (2) tirer.

(faibles coûts vers les comportements dangereux) ainsi que la détection d'erreurs commises par des débutants dans le cadre d'un entraînement (faibles coûts sur les transitions non modélisées dans la bibliothèque de plans).

Une autre approche de reconnaissance de plan hybride bien connue est l'algorithme PHATT (*Probabilistic Hostile Agent Task Tracker*) [36, 42]. Cet algorithme est intéressant parce qu'il est basé sur l'exécution des plans, modélisant ainsi la reconnaissance de plan comme un processus évoluant dans le temps. Il permet de reconnaître, en direct, plusieurs plans exécutés simultanément de façon entrelacée, ce qui est nécessaire dans notre domaine d'application. Cette méthode règle aussi le problème de classement des hypothèses explicatives grâce à un modèle probabiliste. Puisque notre algorithme se base sur PHATT, ce dernier sera discuté en détail dans le chapitre 3.

2.4 Approches génératives

Baker et al. [10] ainsi que Ramírez et Geffner [63, 64, 65] ont récemment commencé à étudier une nouvelle catégorie d'approches pour la reconnaissance de plan. Ces approches ont la particularité de n'utiliser qu'un domaine de planification en entrée (c'est-à-dire les actions, leurs préconditions, leurs postconditions et leur coût, l'état initial et les buts possibles) au lieu d'une bibliothèque, souvent supposée exhaustive bien que cela ne soit pas vrai en pratique, des plans que l'agent observé peut choisir de suivre. Ces méthodes reposent sur le *principe de rationalité* [10, 57], c'est-à-dire que l'agent observé planifie de façon approximativement rationnelle pour atteindre ses buts, étant données ses croyances sur l'état du monde. Avec cette hypothèse, les approches génératives de reconnaissance de plan simulent le processus mental de l'agent observé, puis comparent la séquence d'observations au comportement attendu de l'agent.

Pour comprendre l'intuition derrière ce type d'approches, reprenons l'exemple de Ramírez et Geffner et considérons la figure 2.4, où un agent commençant à la position **I** désire atteindre une des positions **A**–**F**. L'agent peut se déplacer en ligne droite (au coût de 1) ou en diagonale (au coût de $\sqrt{2}$). Les flèches pleines indiquent le chemin emprunté par l'agent pour atteindre son objectif **E**. Après 3 déplacements,

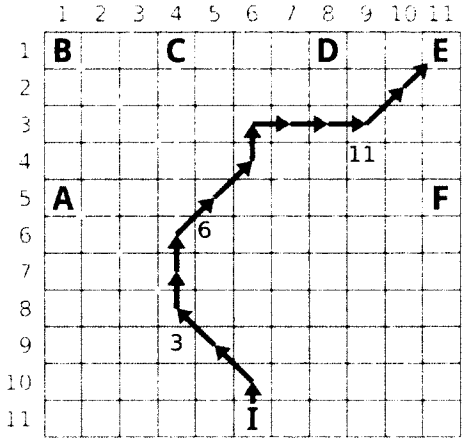


Figure 2.4 – Marche dans une grille [64]

l'agent semble se diriger vers les positions **A**, **B** ou **C**. En effet, les chemins qui seraient empruntés par un agent *rationnel* pour se diriger vers ces positions partagent le même préfixe que celui choisi par l'agent (voir les flèches pointillées). Lorsque l'agent tourne avec son 6^e déplacement, les positions **D** et **E** semblent alors être les plus probables, et après le 11^e déplacement, **E** et **F** semblent être les bonnes destinations de l'agent.

Les approches génératives (ou encore les approches par *planification inverse*) sont donc capables de reconnaître les buts d'un agent observé même si les actions de ce dernier ne correspondent pas tout à fait aux comportements optimaux (plus ou moins équivalents aux plans spécifiés dans les bibliothèques des approches non génératives), et ce, sans traitement supplémentaire ou particulier. La seule hypothèse nécessaire pour les méthodes de cette famille est que le processus de prise de décision de l'agent observé soit connu et qu'il puisse être recréé, que ce soit un planificateur classique [63, 64], un processus de décision de Markov (MDP) [10] ou un MDP partiellement observable (POMDP) [65]. La même idée devrait pouvoir s'appliquer aux autres algorithmes de prise de décision. Dans tous les cas, l'extensibilité de ces approches reste à démontrer dans des applications réelles, puisque le coût du processus de reconnaissance de plan sera au moins aussi coûteux que celui du processus de planification simulé.

2.5 Provocation de l'adversaire

Dans les domaines hautement dynamiques comme les jeux de stratégie en temps réel où l'observateur doit aussi constamment ajuster ses décisions pour tenir compte des actions de l'agent observé, celui-ci sera souvent confronté à des situations où les actions de l'agent observé sont ambiguës. D'un côté, l'observateur ne peut pas attendre que l'agent observé dévoile ses intentions clairement et précisément (en particulier s'il s'agit d'un adversaire). De l'autre, l'observateur ne peut pas toujours se permettre de tirer des conclusions hâtives sur les intentions de l'agent observé : les conséquences d'une mauvaise décision suite à une reconnaissance erronée des plans de l'agent observé pourraient être désastreuses. Dans un jeu RTS, cela pourrait entraîner la défaite de l'observateur ; dans un contexte d'opérations militaires, cela pourrait causer des morts inutiles ou des dommages, directs ou collatéraux.

Tambe et Rosenbloom [81] ont été les premiers à aborder le problème de provocation de l'adversaire. Leur algorithme de reconnaissance de plan ne garde toujours qu'une seule hypothèse, la plus vraisemblable, et l'ajuste si le comportement de l'agent observé devient incohérent avec l'hypothèse. Les auteurs proposent aussi deux méthodes de réduction de l'ambiguïté sur les intentions de l'agent observé, une passive et une active. La méthode passive considère simplement qu'un agent hostile, comme c'est le cas dans leur domaine de combat aérien, risque d'exécuter les comportements les plus dangereux pour l'observateur. Cette interprétation pessimiste des intentions de l'agent observé correspond essentiellement au pire scénario possible, ce qui n'est pas toujours approprié ; cette méthode est donc utilisée avec parcimonie. La méthode active, quant à elle, consiste à exécuter une manœuvre, choisie parmi un ensemble fourni par un expert, qui aura comme effet de forcer l'agent observé à réagir : la réaction devrait permettre de désambiguïser l'intention de l'agent observé. Bien que l'algorithme ne traite pas ce problème, Tambe et Rosenbloom reconnaissent qu'une telle action de provocation puisse interférer avec les buts de l'observateur et que cette interaction est difficile à résoudre. De plus, choisir une seule hypothèse et ignorer les autres n'est pas vraiment adéquat dans un domaine où le temps de réponse pour réagir aux actions de l'adversaire est très court (voir le chapitre 1) et les mêmes problèmes rencontrés avec les approches symboliques refont surface.

2.5. Provocation de l'adversaire

La notion d'exécution proactive d'actions pour désambiguïser des hypothèses sur les intentions d'un autre agent a aussi été explorée par Schrempf et al. [73]. Dans cette approche, un robot collaborant avec un humain doit comprendre l'intention de ce dernier et l'aider à accomplir ses tâches. Lorsque le robot est incapable de comprendre l'intention de l'humain, celui-ci choisit une des hypothèses sur l'intention de l'humain et continue à exécuter des actions répondant à cette intention. Le robot observe alors le comportement de l'humain : s'il devient évident que le robot s'était trompé, il rectifie le tir en exécutant des tâches répondant à l'intention qu'il croit la plus plausible ; sinon, le robot continue d'exécuter des actions servant à aider l'humain à atteindre ses objectifs comme si de rien n'était. Ce dernier ne devrait s'apercevoir de rien et pourrait croire que le robot arrive sans problème à comprendre son intention.

Enfin, l'idée d'exécuter des actions dans l'environnement afin d'obtenir davantage d'information n'est pas nouvelle en intelligence artificielle. En effet, comme l'ont remarqué Russell et Norvig [67], l'information a une grande valeur : des politiques optimales pour des processus de décision de Markov partiellement observables (POMDP), par exemple, contiendront souvent des actions de cueillette d'information, puisque certains états de croyance (*belief states*) peuvent avoir une utilité inférieure à celle d'autres états s'ils ne contiennent pas suffisamment d'information pour prendre une décision éclairée. En robotique mobile, un robot ayant des capteurs bruités ou imparfaits aura parfois de la difficulté à se localiser. Une technique de localisation active [30] peut aider à améliorer l'estimation de la position du robot et le niveau de confiance associé à cette estimation.

Chapitre 3

Algorithme de reconnaissance de plan proposé

Dans ce chapitre, nous décrivons l'algorithme de reconnaissance de plan que nous avons conçu durant ce travail de recherche (PROBE, *Provocation for the Recognition of Opponent BEhaviours*). Comme nous l'avons évoqué en introduction, les humains formulent des hypothèses sur les plans et les buts des autres personnes afin de comprendre leur intention [72]. Notre algorithme, basé sur PHATT [36, 42], consiste aussi à générer des hypothèses expliquant les actions observées. Nous proposons également une technique permettant de trouver le meilleur moyen de provoquer l'adversaire pour observer sa réaction et ainsi désambiguïser les hypothèses sur ses intentions.

Nous commençons donc par présenter le formalisme de modélisation des plans constituant la bibliothèque de plans pouvant être reconnus. Ensuite, nous illustrons le fonctionnement de l'algorithme de reconnaissance de plan de façon intuitive. Puis, nous détaillons l'algorithme de génération des hypothèses servant à expliquer la séquence d'observations obtenues en direct. Nous formalisons par la suite le modèle probabiliste de l'algorithme, qui permet de classer les hypothèses par ordre de vraisemblance. Enfin, nous expliquons comment choisir la meilleure façon de provoquer une réaction chez l'adversaire. La provocation est choisie de façon à ce que les réactions potentielles donnent des indices à l'observateur quant à la véritable intention de l'adversaire. Le chapitre conclut par une discussion analysant les avantages et les inconvénients de l'algorithme, en particulier pour son application dans notre domaine.

3.1 Modélisation des plans de la bibliothèque

Les plans modélisés dans la bibliothèque de plans utilisée par l'algorithme PHATT suivent un formalisme d'arbres « ET/OU » partiellement ordonnés, semblables aux réseaux de tâches hiérarchiques (*Hierarchical Task Networks*, HTN) [39], comme illustré dans la figure 3.1. Cependant, afin de mieux capturer la nature réactive des compor-

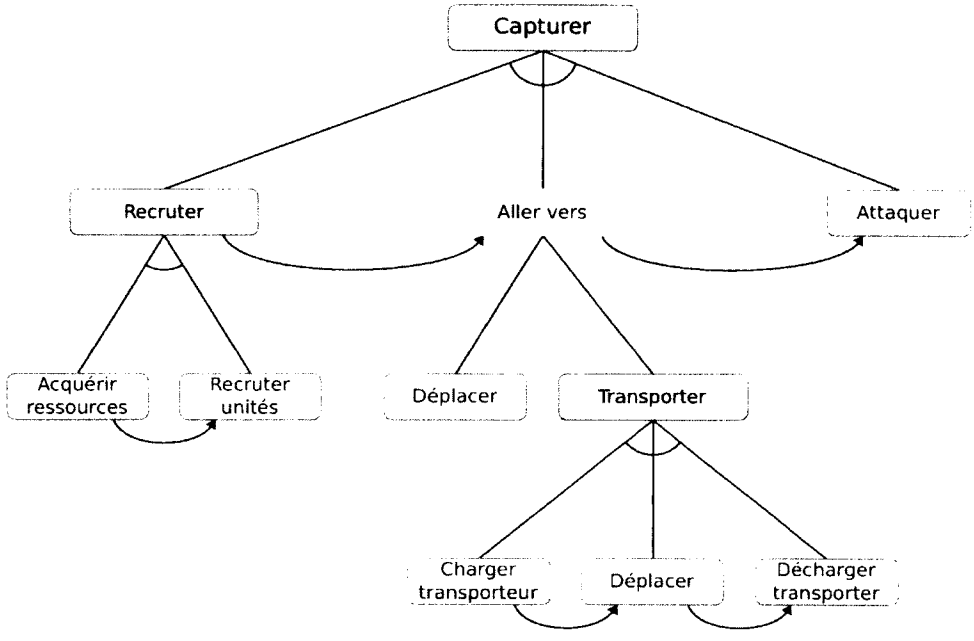


Figure 3.1 – Plan de capture d'une position modélisé par un arbre « ET/OU »

tements de l'adversaire dans un jeu RTS, nous utilisons une représentation différente des arbres « ET/OU » pour modéliser les plans tactiques constituant la bibliothèque de plans. Comme l'ont soulevé De Giacomo et al. [28], un obstacle important empêchant l'intégration de techniques d'intelligence artificielle dans des systèmes réels est la disparité entre la représentation des comportements dans les algorithmes de planification et la représentation des comportements dans les langages de programmation d'agents. La majorité des agents intelligents dans les jeux vidéo sont implémentés à l'aide de concepts basés sur les machines à états finis [62]. Par conséquent, nous avons choisi ce formalisme pour la modélisation de nos comportements, malgré la perte d'ex-

pressivité par rapport aux arbres « ET/OU » [1]. La syntaxe et la sémantique de nos machines à états finis sont inspirées des automates temporisés (*timed automata*) et des machines de Moore dans les méthodes formelles [9, 59].

Définition 3.1. Un *comportement* est un uplet $\beta = (S, A, E, C, s_0, T, \xi)$, où :

- S est un ensemble fini d'états ;
- A est un ensemble fini d'actions ;
- E est un ensemble fini d'événements ;
- C est un ensemble de variables d'horloge ;
- $s_0 \in S$ est l'état initial du comportement ;
- $T : S \times E \rightarrow S$ est la fonction de transition ; et
- $\xi : S \rightarrow A$ est une fonction donnant l'action exécutée dans un état.

L'ensemble des événements auxquels l'agent exécutant ce comportement peut réagir à partir d'un état est donné par la fonction $events : S \rightarrow 2^E$.

Le nom d'un comportement indique le but poursuivi et la machine à états finis sous-jacente donne le plan pour atteindre ce but, comme c'est le cas pour les arbres « ET/OU » dans PHATT. Plusieurs comportements peuvent partager le même nom, reflétant alors les différentes façons d'accomplir un même objectif. Un agent peut bien sûr exécuter plusieurs comportements en parallèle de façon entrelacée ; cette caractéristique est prise en compte par l'algorithme de reconnaissance de plan. La séquence d'observations utilisée par l'algorithme correspond à une séquence d'actions primitives, instanciées avec les bonnes valeurs associées aux paramètres.

Les actions primitives peuvent être paramétrisées (un paramètre peut être global au comportement ou local à un état et aux transitions associées) et sont exécutées lorsque l'agent entre dans un état. Pour pouvoir emprunter une transition et changer d'état, l'événement associé doit être activé. Un événement est une condition booléenne portant sur des propositions logiques de l'état du monde et sur des variables d'horloge. Les propositions logiques sont déclarées à l'extérieur des machines à états finis et permettent de spécifier des faits à propos de la situation présente (par exemple, un groupe

1. Dans sa définition la plus simple, une machine à états finis décrit un langage régulier, alors qu'un HTN décrit un langage hors contexte. Par exemple, les machines à états finis ne peuvent modéliser de comportements de type $a^n b^n$, alors que les arbres « ET/OU » le peuvent. Il est toutefois possible de définir des concepts de machines à états finis étendues, alors bien plus expressifs.

d'unités ennemies y a été aperçu par une unité x : *enemies-sighted*$(x, y) \leftarrow true$). Les variables d'horloge sont utilisées pour spécifier des contraintes temporelles sur les états (par exemple, un état doit être quitté après i unités de temps) ou sur les transitions (par exemple, une transition est activée entre j et k unités de temps). Une transition est *activée* lorsque l'événement associé est vrai et que les contraintes temporelles sont vérifiées.

Étant donnée une machine à états finis modélisant le comportement d'un agent, certains événements font référence à des propositions pouvant être modifiées par un autre agent. Si un agent peut influencer la véracité d'un événement sur une transition ayant comme origine l'état courant, cet événement est dit *contrôlable* ; sinon, il est dit *incontrôlable*. Inversement, si un événement est influencé par un autre agent, nous disons que celui-ci peut être *provoqué* par cet autre agent. Puisque nous cherchons à modéliser les comportements de l'adversaire, nous nous intéresserons particulièrement, dans la section 3.5, aux événements auquel l'adversaire réagit pouvant être provoqués par l'observateur.

La figure 3.2 montre la représentation graphique d'un comportement modélisé avec ce formalisme. L'état initial $s_0 \in S$ est pointé par un cercle plein. Chaque état s est étiqueté par l'action $\xi(s) \in A$ qui y est exécutée, avec les paramètres requis. De façon similaire, les transitions sont étiquetées par l'événement auquel l'agent réagit en empruntant la transition. Les transitions sur des événements que l'autre agent (ici, l'observateur) peut provoquer sont pointillées. Pour simplifier la notation, les variables d'horloges et les contraintes temporelles ont été omises. Par exemple, la

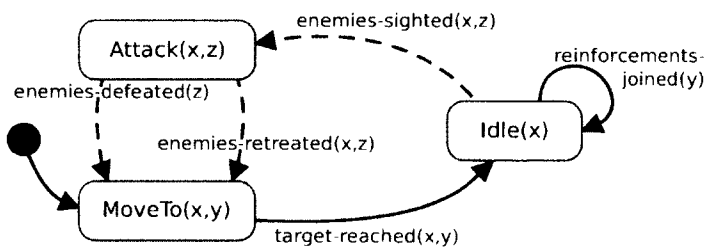


Figure 3.2 – Comportement de confinement – *Contain*(x, y)

figure 3.2 modélise le comportement d'un adversaire voulant confiner un autre agent à l'intérieur de sa propre base. Ainsi, l'adversaire commence par se rendre à cet

endroit (*MoveTo(x,y)*), après quoi il attend (*Idle(x)*) d'observer des ennemis (*enemies-sighted(x,z)*). Dès que cet événement (contrôlable du point de vue de l'autre agent, et pouvant donc être provoqué) survient, l'adversaire attaque les ennemis tentant de s'enfuir (*Attack(x,z)*).

3.2 Intuition derrière l'algorithme

Comme l'algorithme PHATT, notre algorithme traite le problème de reconnaissance de plan comme un processus d'abduction logique [25] : il tente d'inférer les causes (les plans) étant donnés les effets (les actions observées). De plus, l'algorithme de reconnaissance de plan est basé sur un modèle d'exécution des plans. Cela signifie que le processus de reconnaissance est vu comme un mécanisme évoluant dans le temps, au fur et à mesure que les actions constituant ces plans sont observées. Par exemple, si des actions pouvant appartenir à un plan d'*Attaque* ou à un plan de *Défense* sont observées, l'algorithme de reconnaissance de plan devrait conclure que ces deux hypothèses sont valables. Cependant, dès que des actions qui n'appartiennent qu'au plan d'*Attaque* sont observées, l'algorithme devrait plutôt privilégier cette hypothèse comme étant la plus vraisemblable ; si l'hypothèse du plan de *Défense* était la bonne, des actions appartenant à ce plan devraient être observées tôt ou tard.

À cette fin, l'algorithme de reconnaissance de plan (tout comme le faisait PHATT) déroule à la volée un modèle de Markov caché (*Hidden Markov Model*, HMM) afin de conserver la suite des hypothèses cohérentes avec la séquence des observations des actions de l'agent. En d'autre termes, un état caché du HMM est un ensemble de plans que l'agent observé pourrait être en train d'exécuter à un moment donné avec l'état d'exécution de ces plans : ce sont les *hypothèses*, que nous formaliserons dans un instant. Les variables d'observation du HMM correspondent évidemment aux observations des actions de l'agent observé. Ce processus est illustré dans la figure 3.3 : l'état initial d'exécution des plans est calculé à partir des états initiaux des comportements de la bibliothèque, et l'ensemble des hypothèses valides est calculé après chaque nouvelle observation afin de mettre à jour le modèle de reconnaissance.

Définition 3.2. Une *trace* dans un comportement $\beta = (S, A, E, C, s_0, T, \xi)$ est une séquence $v = \langle s_0, t_0, e_0, \ldots, s_i, t_i, e_i, s_{i+1}, t_{i+1}, \ldots, e_{n-1}, s_n, t_n \rangle$ indiquant la suite des

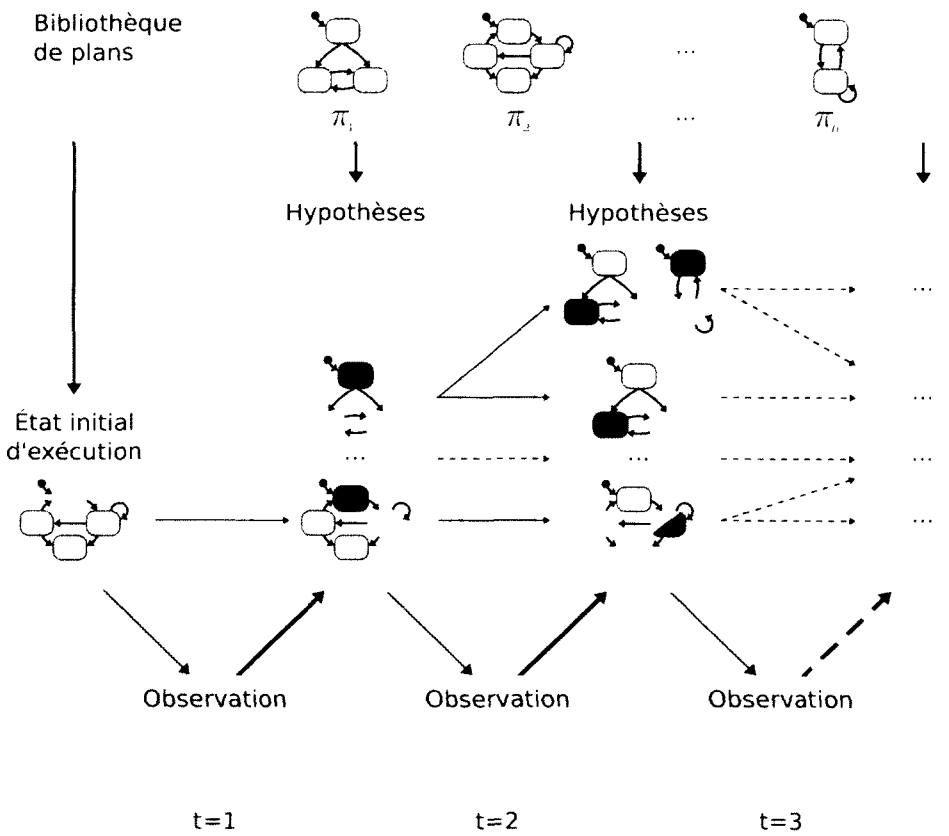


Figure 3.3 – Processus de reconnaissance de plan avec un modèle de Markov caché

états $s_i \in S$ visités aux temps t_i et des événements $e_j \in E$ des transitions associées qui ont été empruntées pour passer de l'état initial s_0 à l'état courant s_n.

Un exemple graphique d'une trace $\upsilon = \langle s_0, t_0, e_0, s_1, t_1, e_1, s_2, t_2 \rangle$ est illustré dans la figure 3.4. Les états et les événements du comportement sont étiquetés dans la trace avec s_i et e_j, respectivement, selon l'ordre dans lequel les états ont été visités et les transitions empruntées.

Définition 3.3. Une *hypothèse* est une paire $\eta = (B, \Psi)$, où :

- B est un ensemble de comportements présentement exécutés par l'agent ; et
- Ψ est une fonction qui associe un comportement $\beta \in B$ à la trace $\upsilon = \Psi(\beta)$ des états visités et des transitions empruntées.

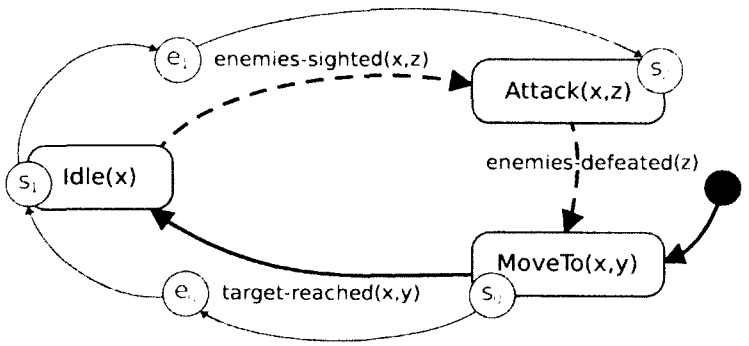


Figure 3.4 – Exemple graphique d'une trace dans un comportement

L'état courant d'un comportement $\beta = (S, A, E, C, s_0, T, \xi) \in B$ sous l'hypothèse η peut être obtenu par la fonction $curr : B \rightarrow S$ qui retourne le dernier état s_n dans la trace $\psi = \Psi(\beta) = \langle s_0, t_0, e_0, \ldots, s_i, t_i, e_i, s_{i+1}, t_{i+1}, \ldots, e_{n-1}, s_n, t_n \rangle$.

Si un agent choisit, sous une hypothèse η, d'exécuter plusieurs instances du même comportement (par exemple, attaquer plusieurs cibles différentes), chacune de celles-ci sera incluse dans l'ensemble B.

À partir d'une hypothèse $\eta = (B, \Psi)$ et de la fonction $curr$, il est facile de calculer l'ensemble des actions activées parmi lesquelles l'agent pourrait choisir sa prochaine action à exécuter :

$$actions(B, curr) = \bigcup_{\substack{\beta \in B, \\ e \in events(curr(\beta))}} \xi\left(T(curr(\beta), e)\right) \tag{3.1}$$

Notons enfin que l'agent pourrait également décider de commencer un nouveau comportement en exécutant l'action dans l'état initial de ce comportement, cas non inclus dans l'équation (3.1), mais bien sûr traité dans l'algorithme.

Ainsi, à partir des hypothèses dans un état du HMM, il est possible de prédire quelle sera la prochaine action de l'agent à être observée. En effet, étant donnée la supposition d'exhaustivité de la bibliothèque de plans et un environnement parfaitement et entièrement observable, la prochaine action de l'agent à être observée se trouvera forcément dans l'ensemble $actions(B, curr)$ d'au moins une des hypothèses ou encore dans l'état initial d'un des comportements de la bibliothèque.

3.3 Génération des hypothèses

L'algorithme de génération d'hypothèses est donné dans l'algorithme 3.1. Les paramètres en entrée sont l'ensemble des hypothèses présentement considérées (H), la séquence d'observations ($\mathcal{O}$), la bibliothèque de comportements (Π), la nouvelle observation (α) et le temps auquel elle a été obtenue (t). Avant le premier appel, l'ensemble des hypothèses H est initialisé avec une seule hypothèse vide, $(\varnothing, \langle\rangle)$, signifiant que l'agent n'a pas encore commencé à exécuter de plan. L'algorithme retourne en sortie l'ensemble des hypothèses nouvellement générées ainsi que la séquence d'observations concaténée avec la nouvelle observation.

Algorithme 3.1 Génération des hypothèses

```
1.  GénérerHypothèses(H, O, Π, α, t)
2.    H ← ÉliminerHypothèsesExpirées(H, t)
3.    H' ← ∅
4.    for each η = (B, Ψ) ∈ H

5.      for each β ∈ B
6.        if α ∈ ActionsActivées(β, curr(β), t) then
7.          for each e, s such that s = T(curr(β), e) ∧ α = ξ(s)
8.            (B', Ψ') ← (B, Ψ)
9.            Ψ'(β) ← Ψ'(β) + ⟨e, s, t⟩
10.           H' ← H' ∪ (B', Ψ')

11.     for each β ∈ Π
12.       if α = ξ(s0) then
13.         (B', Ψ') ← (B, Ψ)
14.         B' ← B ∪ β
15.         Ψ'(β) ← ⟨s0, t⟩
16.         H' ← H' ∪ (B', Ψ')

17.   O' ← O + ⟨α⟩
18.   return (H', O')
```

L'algorithme commence par éliminer de l'ensemble des hypothèses présentement considérées celles qui ont expiré, parce que certaines de leurs contraintes temporelles sont violées étant donné le temps t auquel la nouvelle observation α a été obtenue.

Ensuite, la boucle débutant à la ligne 4 itère sur toutes les hypothèses existantes afin de les mettre à jour de deux façons différentes.

Étendre une comportement existant Intuitivement, la première façon consiste à ajouter à la trace, pour chaque comportement de l'hypothèse dans lequel l'action α est activée, la transition empruntée pour exécuter cette action. Ainsi, l'algorithme itère d'abord sur tous les comportements de l'hypothèse (ligne 5), puis sur chaque transition de ce comportement telle que l'état courant est l'état d'origine de la transition et que l'état suivant est étiqueté avec l'action observée (ligne 7). La fonction ACTIONSACTIVÉES() (semblable à l'équation (3.1)) appelée à la ligne 6 s'assure que l'action observée est activée en comparant le temps t d'observation de l'action α aux contraintes temporelles sur les variables d'horloge C du comportement β. Une nouvelle hypothèse est générée chaque fois que ces conditions sont vérifiées en ajoutant, à la ligne 9, la transition empruntée (l'événement et l'état suivant) à la trace du comportement dans la nouvelle hypothèse. L'hypothèse est finalement ajoutée à l'ensemble des nouvelles hypothèses H'. La figure 3.5 illustre ce moyen de produire une hypothèse.

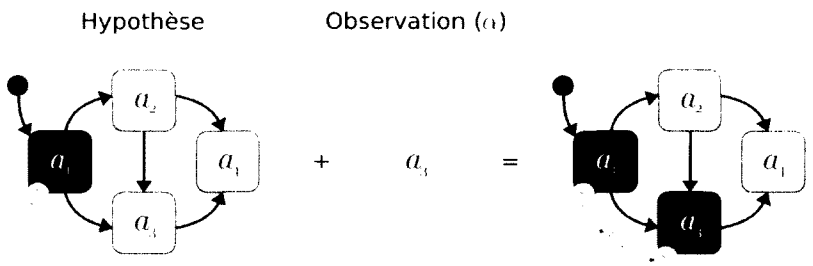


Figure 3.5 – Génération d'une hypothèse en étendant un comportement existant

Ajouter un nouveau comportement La deuxième façon de générer une hypothèse consiste à ajouter un nouveau comportement de la bibliothèque à l'hypothèse. Pour ce faire, l'algorithme vérifie si la nouvelle observation correspond à l'action exécutée dans l'état initial de chacun des comportements de la bibliothèque de plans Π. Si un comportement satisfaisant cette condition est trouvé, il est ajouté à l'hypothèse (ligne 14) et la trace de ce comportement dans l'hypothèse est initialisée uniquement

avec son état initial (ligne 15). Enfin, la nouvelle hypothèse ainsi générée est ajoutée à l'ensemble des nouvelles hypothèses H'. La figure 3.6 illustre cette façon de générer une nouvelle hypothèse.

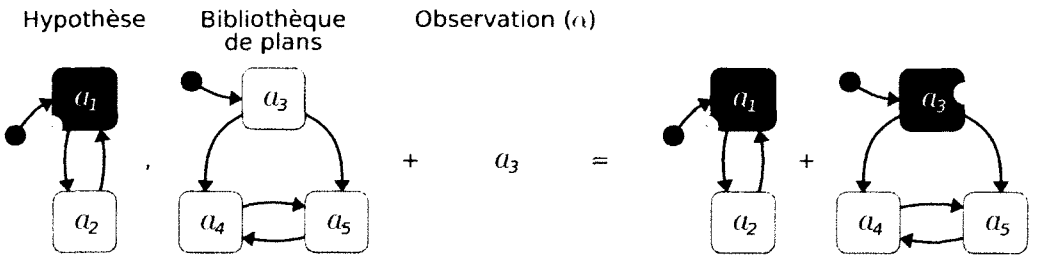


Figure 3.6 - Génération d'une hypothèse en ajoutant un nouveau comportement

Il est important de noter qu'une même hypothèse peut être mise à jour à plusieurs reprises, selon l'une, l'autre ou même selon les deux méthodes de génération d'une hypothèse. Selon la structure des plans dans la bibliothèque, le nombre d'hypothèses générées d'une étape à l'autre peut grandir de façon exponentielle dans le pire cas [33].

3.4 Modèle probabiliste

Une fois les hypothèses générées, il ne reste qu'à établir une distribution de probabilités sur celles-ci, étant donnée la séquence d'observations. Ceci permet de régler le problème rencontré par les approches de reconnaissance de plan purement logiques, où toutes les hypothèses sont aussi valables les unes que les autres. Nous commençons donc avec la formule de Bayes pour obtenir l'équation suivante :

$$P(\eta \mid \mathcal{O}) = \frac{P(\eta \wedge \mathcal{O})}{P(\mathcal{O})} = \frac{P(\eta)P(\mathcal{O} \mid \eta)}{P(\mathcal{O})} \tag{3.2}$$

Le dénominateur de l'équation (3.2) peut être ignoré pour l'instant puisqu'il est constant pour toutes les hypothèses et peut donc être calculé en faisant la somme de tous les numérateurs afin de normaliser la distribution de probabilités.

La probabilité d'une hypothèse, $P(\eta)$, est scindée en deux sous-facteurs indiquant, respectivement, la probabilité *a priori* de choisir les comportements et la façon dont

ceux-ci sont exécutés :

$$P(\eta) = P(B) \times \prod_{\substack{(s_i, e_i, s_{i+1}) \\ \in \Psi(\beta)}} P(s_{i+1}, e_i \mid s_i) \tag{3.3}$$

avec $P(B) \propto \prod_{\beta \in B} \varphi(\beta)$ [2], et où (s_i, e_i, s_{1+i}) dénote chaque transition telle que $s_{i+1} = T(s_i, e_i)$ dans la trace $v = \Psi(\beta)$. Par exemple, une trace $v = \langle s_0, t_0, e_0, s_1, t_1, e_1, s_2, t_2 \rangle$ contient 2 transitions, (s_0, e_0, s_1) et (s_1, e_1, s_2). Chacun des facteurs $\varphi(\beta)$ a une valeur dans l'intervalle $[0, 1]$ et correspond à la préférence supposée que l'agent observé choisisse le comportement β pour constituer son but. Toutes ces valeurs doivent être spécifiées *a priori* avec la bibliothèque de plans. Bien que cela ne soit pas nécessaire, une contrainte de sommation à 1 a également été imposée sur le choix des valeurs de facteurs (comme si elles correspondaient à des probabilités). Puisque ces paramètres sont difficiles à obtenir, des distributions uniformes sont souvent utilisées en pratique, mais des méthodes d'apprentissage ou d'optimisation pourraient servir à les estimer.

Le second facteur du numérateur de l'équation (3.2) calcule la probabilité que chaque observation o_i dans la séquence $\mathcal{O}$ ait été celle choisie par l'agent, parmi toutes les actions possibles (l'ensemble des actions activées au temps $i - 1$). La probabilité de la séquence d'observations sachant l'hypothèse, $P(\mathcal{O} \mid \eta)$, est donc calculée en trouvant l'ensemble des actions activées à l'aide de l'équation (3.1) [3], puis en donnant la probabilité que chaque observation $o_i \in \mathcal{O}$ soit celle choisie parmi cet ensemble :

$$P(\mathcal{O} \mid \eta) = \prod_{i=1}^{n} P(o_i \mid actions(B, curr_{\bar{o}_{i-1}})) \tag{3.4}$$

où $curr_{\bar{o}_i}$ dénote la fonction *curr* lorsque la séquence d'observations $\mathcal{O}$ était égale au préfixe $\bar{o}_i = \langle o_1, \ldots, o_i \rangle$. Notons que $actions(B, curr_{\bar{o}_0})$ correspond à l'ensemble des actions initiales des comportements dans l'hypothèse. Remarquons aussi que l'en-

2. Nous supposons que le nombre de comportements pouvant faire partie d'une hypothèse est borné par un très grand nombre, qui est constant. Ainsi, la constante de normalisation dans $P(B)$ ne dépend pas de B et n'a donc pas à être évaluée lors du calcul des probabilités *a posteriori*.

3. Afin de simplifier l'écriture, nous considérons qu'il existe, pour chaque comportement dans la bibliothèque, une transition artificielle vers l'état initial de ce comportement. Ainsi, l'ensemble $actions(B, curr_{\bar{o}_i})$ contient aussi les actions initiales des comportements faisant partie de l'hypothèse η, mais pas encore commencés par l'agent au temps i.

semble des actions activées après la n^e observation n'est pas considéré dans l'équation, puisqu'aucune action de cet ensemble n'a encore été observée. Par conséquent, l'historique des hypothèses doit être conservé en mémoire. Cependant, afin d'alléger l'utilisation de la mémoire vive, les traces des comportements sont annotées avec les temps auxquels les observations ont été obtenues, de façon à pouvoir calculer les fonctions $curr_{\bar{o}_i}$ à la volée. Notons que la probabilité conditionnelle est elle aussi souvent approximée par une distribution uniforme en pratique.

En combinant les équations (3.2), (3.3) et (3.4), nous obtenons l'équation (3.5), qui donne le modèle probabiliste des hypothèses dans notre algorithme :

$$P(\eta \mid \mathcal{O}) = \frac{\prod_{\beta \in B} \varphi(\beta) \times \prod_{\substack{(s_i, e_i, s_{i+1}) \\ \in \Psi(\beta)}} P(s_{i+1}, e_i \mid s_i) \prod_{i=1}^{n} P(o_i \mid actions(B, curr_{\bar{o}_{i-1}}))}{P(\mathcal{O})} \tag{3.5}$$

Enfin, nous pouvons calculer la probabilité d'un but G[4] étant données les observations en faisant la somme des probabilités des hypothèses ayant ce but :

$$P(G \mid \mathcal{O}) = \sum_{\eta_G \in H} P(\eta_G \mid \mathcal{O}) \tag{3.6}$$

3.5 Provoquer une réaction

Dans cette section, nous étudions une méthode heuristique pour aider l'algorithme de reconnaissance de plan à désambiguïser l'intention de l'adversaire. Cette méthode consiste à observer de quelle façon l'adversaire réagit à une provocation bien orchestrée [15, 16, 17].

Rappelons-nous que certains des événements dans les comportements de l'adversaire peuvent être provoqués par l'observateur, c'est-à-dire que les conditions associées à ces événements sont modifiées par des actions exécutées par l'observateur. Provoquer un événement peut entraîner une réaction chez l'adversaire : celui-ci pourrait décider de suivre la transition activée par l'événement. L'intuition est qu'en réagissant à la provocation, l'adversaire pourrait révéler involontairement son intention en exécutant

4. Un but correspond à un ensemble de comportements à exécuter.

une action, observée ensuite par l'observateur, appartenant à un comportement qui permet d'atteindre le but souhaité.

Pour parvenir à provoquer l'adversaire, nous avons besoin d'un mécanisme pour décider quand provoquer quel événement. Il s'agit en fait d'un problème de planification, que nous tentons cependant de résoudre de façon heuristique. Une approche plus complète tiendrait aussi compte de l'effet à plus long terme d'une provocation sur les propres buts de l'observateur (par exemple, les actions exécutées pour provoquer l'adversaire pourraient lui révéler la position de l'observateur et le rendre vulnérable). Nous y reviendrons dans la discussion à la fin de ce chapitre.

Plusieurs mesures pourraient être utilisées pour qualifier et sélectionner un événement à provoquer. Par exemple, nous pourrions choisir l'événement qui confirmerait l'hypothèse la plus probable, ou nous pourrions éviter les événements qui mènent à des actions dangereuses pour l'observateur.

Nous avons plutôt choisi une mesure basée sur la désambiguïsation espérée de l'ensemble des hypothèses expliquant la séquence des actions de l'adversaire observées jusqu'à maintenant. En d'autres termes, nous calculons, pour chaque événement pouvant être provoqué, l'espérance de la réduction de l'incertitude associée à l'intention de l'adversaire, sachant de quelle façon il pourrait réagir à une provocation (c'est-à-dire en examinant les transitions associées à ces événements dans l'ensemble d'hypothèses). En théorie de l'information, l'entropie est fréquemment employée pour mesurer l'incertitude associée à une variable aléatoire [27]. En utilisant l'entropie comme mesure d'incertitude sur l'ensemble des hypothèses expliquant la séquence d'observations, nous pouvons déterminer quelle provocation permettrait de réduire au maximum l'ambiguïté sur l'intention de l'adversaire.

Soit $H = \{\eta_1, \ldots, \eta_n\}$, l'ensemble des hypothèses expliquant les observations. L'entropie $\mathcal{H}$ de cet ensemble étant données les observations est définie comme suit :

$$\mathcal{H}(H \mid \mathcal{O}) = -\sum_{i=1}^{n} P(\eta_i \mid \mathcal{O}) \log_2 P(\eta_i \mid \mathcal{O})$$

où $P(\eta_i \mid \mathcal{O})$ est bien sûr donnée par l'équation (3.5).

En sélectionnant un événement pouvant être provoqué et en simulant les réactions possibles comme si l'événement était effectivement survenu, nous obtenons un nou-

vel ensemble d'hypothèses. Nous pouvons ensuite comparer l'entropie de ce nouvel ensemble d'hypothèses avec celle du précédent pour déterminer le gain ou la réduction d'incertitude suite à la provocation de cet événement. Il est important de noter que le nouvel ensemble d'hypothèses n'a pas nécessairement la même cardinalité que le précédent : certaines hypothèses peuvent avoir été éliminées alors que d'autres peuvent avoir été générées. Afin de pouvoir comparer correctement l'entropie d'ensembles d'hypothèses de cardinalités différentes, nous normalisons l'entropie en la divisant par $\log_2 |H|$, ramenant ainsi la valeur entre 0 et 1.

Pour formaliser, l'ensemble des événements pouvant être provoqués par l'observateur étant donné l'ensemble des hypothèses est noté Φ. Pour chaque événement $\phi \in \Phi$, nous calculons la désambiguïsation espérée de l'intention de l'adversaire :

$$\mathcal{D}(H, \mathcal{O}, \phi) = \frac{\mathcal{H}(H \mid \mathcal{O})}{\log_2(|H|)} - \sum_{\alpha \in A(\phi)} P(\alpha \mid \phi) \frac{\mathcal{H}(H_\alpha \mid \mathcal{O} + \langle \alpha \rangle)}{\log_2(|H_\alpha|)} \tag{3.7}$$

où $A(\phi)$ dénote l'ensemble des actions possibles de l'adversaire en réaction à la provocation de l'événement ϕ, et où H_α dénote l'ensemble des hypothèses suite à l'observation de la réaction α.

Le codomaine de la fonction $\mathcal{D}$ est $[-1, 1]$, chaque extrême marquant une situation où l'ambiguïté sur l'intention de l'adversaire a été exacerbée de la pire façon possible ou complètement enlevée, respectivement. Le meilleur événement à provoquer est donc celui qui maximise la valeur $\mathcal{D}(H, \mathcal{O}, \phi)$. Si cette valeur est négative, il serait sans doute préférable de ne rien faire et d'attendre, plutôt que de risquer d'augmenter l'incertitude sur l'intention de l'adversaire.

Nous pouvons maintenant décrire l'algorithme de reconnaissance de plan avec provocation (algorithme 3.2). L'algorithme prend en entrée la bibliothèque de plans Π et deux seuils γ et δ spécifiant l'entropie minimale et la désambiguïsation espérée minimale, respectivement.

L'ensemble des hypothèses est d'abord initialisé avec une seule hypothèse vide à la ligne 2. Ensuite, la boucle débutant à la ligne 4 attend qu'une action soit observée (ligne 3) pour générer les hypothèses expliquant la séquence d'observations grâce à l'algorithme 3.1[5]. L'entropie de l'intention de l'adversaire est calculée à la ligne 7

5. L'algorithme pourrait facilement être adapté pour sélectionner automatiquement un événement

Algorithme 3.2 Reconnaissance de l'intention de l'adversaire

```
1.  RECONNAÎTREINTENTION(Π, γ, δ)
2.    H ← {(∅, ⟨⟩)}
3.    O ← ⟨⟩
4.    loop
5.      (α, t) ← OBSERVERACTION()
6.      (H, O) ← GÉNÉRERHYPOTHÈSES(H, O, Π, α, t)
7.      entropie ← H(H | O)
8.      if entropie ≥ γ then
9.        (φ, D) ← SÉLECTIONNERÉVÉNEMENT(H, O, Π)
10.       if D ≥ δ then
11.         PROVOQUER(φ)
```

et comparée au seuil γ. Si l'entropie (l'incertitude sur l'intention de l'adversaire) est supérieure ou égale à ce seuil, la désambiguïsation espérée de chaque événement pouvant être provoqué est calculée grâce à une simple application de l'équation (3.7) ; la fonction SÉLECTIONNERÉVÉNEMENT() retourne l'événement ϕ produisant la valeur maximale D. Finalement, si la désambiguïsation espérée $D = \mathcal{D}(H, \mathcal{O}, \phi)$ est supérieure ou égale au seuil δ, l'événement est provoqué. La provocation de l'événement serait, par exemple, donnée comme but au module de prise de décision de l'observateur afin de produire un plan menant à un état où l'événement est survenu (c'est-à-dire que les conditions associées à l'événement sont vérifiées). Dans notre implémentation actuelle, l'agent observé réagit systématiquement aux provocations.

Tel que mentionné précédemment, nous ne considérons pour l'instant que la réduction espérée de l'incertitude associée à l'intention de l'adversaire. Cependant, une solution plus complète tiendrait aussi compte du coût du plan de provocation pour choisir le meilleur événement en faisant une combinaison linéaire des deux valeurs :

$$V(H, \mathcal{O}, \phi) = k_1 \cdot \mathcal{D}(H, \mathcal{O}, \phi) + k_2 \cdot \mathcal{C}(\pi_\phi) \tag{3.8}$$

où $\mathcal{C}(\pi_\phi)$ indique le coût du plan π_ϕ de provocation de l'événement ϕ, et où k_1 et k_2 sont des constantes spécifiées par l'utilisateur du système.

à provoquer si aucune action n'est observée pendant une certaine période.

3.6 Discussion

3.6.1 Algorithme de reconnaissance de plan

L'algorithme PHATT, sur lequel se base notre algorithme de reconnaissance de plan, possède plusieurs caractéristiques qui le démarquent des autres approches de reconnaissance de plan. En particulier, il permet, en mettant à jour le modèle de reconnaissance à chaque nouvelle observation, de reconnaître en temps réel les plans de l'agent observé, ce qui en fait un choix très intéressant pour des domaines comme les jeux RTS et l'évaluation de la menace dans la guerre navale, où la compréhension de la situation doit constamment être actualisée pour que la prise de décision s'appuyant sur les résultats de la reconnaissance ne devienne pas obsolète. L'expression « temps réel » est cependant utilisée de façon quelque peu libérale et signifie ici que l'algorithme est capable d'ajuster ses inférences au fur et à mesure que les actions de l'agent sont observées, par opposition aux approches qui nécessitent de prime abord une séquence complète des observations (par exemple, l'algorithme de Sadilek et Kautz [68]). L'algorithme (du moins dans son implémentation actuelle) n'est aucunement soumis à des contraintes de temps réel strictes (*hard*) ou souples (*soft*).

Le problème soulevé plus tôt d'explosion combinatoire du nombre d'hypothèses générées rend difficile l'utilisation à moyen ou à long terme d'un tel algorithme dans une application où la séquence d'observations grandit très rapidement et où le nombre de plans est immense, sans avoir recours à des méthodes heuristiques pour éliminer des hypothèses [79]. En ce qui concerne l'implémentation et l'utilisation pratique de l'algorithme, la grande quantité d'informations à conserver en mémoire cause rapidement des problèmes de dépassement de capacité, et est évidemment exacerbé par la croissance exponentielle du nombre d'hypothèses. L'ajout des contraintes temporelles (par rapport à PHATT) a cependant permis d'atténuer légèrement ces problèmes dans notre domaine d'application en élaguant les hypothèses dont les contraintes temporelles sont violées. Aussi, certaines optimisations ont été implémentées dans l'algorithme, comme par exemple l'historique des ensembles d'actions activées qui n'est calculé que lorsque nécessaire, plutôt que conservé en entier dans chaque hypothèse, afin d'alléger l'utilisation de la mémoire.

Tel que décrit, l'algorithme de reconnaissance de plan suppose que l'environnement

est parfaitement et entièrement observable. Cependant, puisque l'algorithme est fortement inspiré de PHATT, nous croyons pouvoir implémenter sans trop de difficulté l'extension proposée par Geib et Goldman [35] qui permet de résoudre ce problème. En résumé, cette méthode permet de considérer que certaines observations pourraient avoir été manquées pour diverses raisons (capteurs imparfaits, interdiction d'accès à l'information délibérée, etc.) en générant des hypothèses contenant des actions non observées et en ajustant le calcul des probabilités conditionnelles pour tenir compte de la probabilité qu'une action ait été exécutée, mais pas observée.

L'algorithme de reconnaissance de plan est capable de reconnaître plusieurs plans exécutés en parallèle de façon entrelacée et partiellement ordonnée. Il peut aussi reconnaître qu'un même plan puisse avoir été exécuté à plusieurs reprises à des instants différents (par exemple, un plan d'attaque exécuté plusieurs fois durant une partie), ce que bien des algorithmes sont incapables de faire (par exemple, le système de Kautz et Allen [46]). Toutes ces caractéristiques font de cet algorithme un bon choix pour la reconnaissance de plan dans les jeux RTS malgré les problèmes mentionnés plus tôt.

Comme l'ont noté Geib et Goldman [36] ainsi que Pynadath et Wellman [61], les algorithmes de reconnaissance de plan sont très semblables aux algorithmes de reconnaissance de forme ou de motif (*pattern recognition*). En effet, une bibliothèque de plans est en fait un ensemble de motifs d'actions pouvant être observés. La différence est cependant que les plans sont aussi des structures rationnelles, synthétisés par des agents intelligents avec des connaissances et des préférences [61]. De plus, une particularité de PHATT (et donc de notre algorithme) est que la reconnaissance de plan est basée sur un modèle de l'exécution des plans. La reconnaissance de plan est alors un mécanisme évoluant dans le temps, au fur et à mesure que des actions exécutées par un agent rationnel sont observées. Il ne s'agit donc pas simplement d'une collection de données que l'algorithme doit faire correspondre à des motifs dans une bibliothèque de plans. C'est entre autres cette vision de la reconnaissance de plan qui nous a permis d'introduire la notion de provocation de l'adversaire.

Finalement, comme tous les algorithmes de reconnaissance de plan probabilistes, les paramètres fournis *a priori* en entrée sont difficiles à obtenir. Ainsi, les différentes distributions de probabilités utilisées par le modèle probabiliste (par exemple, la probabilité qu'une observation soit choisie parmi un ensemble d'actions activées)

sont souvent approximées par des distributions uniformes en pratique. L'algorithme ne dépend évidemment pas de cette simplification : si la véritable distribution était disponible, elle n'aurait qu'à être insérée dans le calcul de la probabilité *a posteriori* d'une hypothèse étant données les observations. Tous ces paramètres pourraient par exemple être obtenus grâce à un algorithme d'apprentissage automatique analysant un grand nombre de parties préenregistrées, ou même de parties contre un joueur en particulier afin d'obtenir un modèle de cet adversaire [11, 70].

Bien que le choix d'utiliser des machines à états finis plutôt que des arbres « ET/OU » ne change pas le fonctionnement de l'algorithme sur le plan fondamental, l'algorithme de génération d'hypothèses ainsi que les équations du calcul de la probabilité de distribution sur l'ensemble d'hypothèses ont dû être retravaillés pour s'adapter au nouveau formalisme. Dans notre implémentation de l'algorithme, les deux formalismes peuvent être utilisés de façon interchangeable selon les besoins.

3.6.2 Provocation de l'adversaire

Bien que l'idée de provoquer l'adversaire pour observer sa réaction soit relativement indépendante de l'algorithme de reconnaissance de plan sous-jacent, notre méthode utilise un algorithme appartenant à la famille des approches non génératives. La seule connaissance supplémentaire que notre approche exige est la réaction de l'adversaire aux provocations, connaissance qui pourrait sûrement être extraite automatiquement d'un domaine de planification avec des adversaires en analysant les effets et les préconditions des actions des deux joueurs, plutôt que fournie explicitement dans une bibliothèque de plans. Nous croyons donc que notre méthode pourrait s'intégrer à une approche générative de reconnaissance de plan sans grande difficulté.

Notre algorithme de provocation a le même objectif que celui de Tambe et Rosenbloom [81] : réduire l'ambiguïté sur l'intention de l'adversaire en incitant ce dernier à réagir à une provocation. Cependant, notre approche est capable d'extraire de la bibliothèque de plans et de choisir dynamiquement la meilleure façon de provoquer l'adversaire, alors que la méthode de Tambe et Rosenbloom nécessite qu'un expert ait spécifié *a priori* les manœuvres de provocation à choisir selon la situation. Notre méthode est donc plus flexible, puisque le concepteur des plans de la bibliothèque

n'a qu'à indiquer les événements auxquels l'adversaire réagit (et lesquels peuvent être provoqués) sans devoir imaginer toutes les situations possibles et spécifier pour chacune d'elles la meilleure façon de provoquer l'adversaire. Cependant, aucune des deux approches ne considère les effets à plus ou moins long terme de la provocation sur les propres buts de l'observateur.

L'idée de Schrempf et al. [73] d'exécution proactive d'actions pour désambiguïser des hypothèses sur les intentions d'un autre agent est similaire à la nôtre : cependant, l'objectif de cette méthode est de bien choisir une intention (parmi les hypothèses) et d'exécuter des actions y correspondant comme si c'était la bonne, alors que dans notre approche l'objectif est plutôt de choisir une provocation afin de minimiser l'ambiguïté sur l'intention de l'adversaire. Néanmoins, le concept de provocation ou d'exécution d'actions proactives n'est pas limité aux domaines avec des adversaires ou aux domaines de reconnaissance intentionnelle, respectivement, mais bien aux types de reconnaissance de plan où l'agent observé et l'observateur évoluent dans le même environnement et où ils peuvent s'influencer l'un l'autre. Il serait donc sans doute possible d'appliquer leur méthode dans notre domaine et *vice versa*. Par contre, les conséquences d'une mauvaise sélection d'une intention dans la méthode de Schrempf et al. pourraient être catastrophiques dans notre domaine, comme nous l'avons expliqué précédemment : mieux vaut être prudent et s'assurer avec un certain degré de confiance que l'intention de l'adversaire est bien comprise avant d'agir. Ce problème est aussi présent dans la méthode de Tambe et Rosenbloom [81], où l'observateur ne conserve qu'une seule hypothèse et ajuste le tir au besoin.

Comme nous l'avons mentionné plus tôt, notre approche ne considère pour l'instant que la réaction immédiate de l'adversaire, sans réellement tenir compte des effets à long terme sur les objectifs de l'observateur. Ainsi, une provocation (ou la réaction de l'adversaire à cette provocation) pourrait nuire, voire empêcher, l'observateur d'accomplir ses propres buts, tout en permettant cependant de réduire l'ambiguïté sur l'intention de l'adversaire. Ce problème pourrait en partie être résolu en utilisant d'autres facteurs que la réduction de l'incertitude sur l'intention de l'adversaire, comme dans l'équation (3.8). Le coût d'un plan pourrait, par exemple, capturer le fait qu'un plan de provocation mène à un état indésirable pour l'observateur et ainsi de suite. Nous pourrions aussi ne baser le choix d'un événement à provoquer que sur les

k hypothèses les plus probables plutôt que sur l'ensemble au complet. Cela pourrait considérablement réduire le temps de recherche dans les situations où la cardinalité de l'ensemble d'hypothèses est immense.

Dans notre approche, les réactions de l'adversaire aux provocations sont traitées comme toutes les autres observations. Il serait pertinent d'étudier l'impact de faire une analyse particulière des réactions par rapport aux autres observations afin que l'algorithme puisse mieux comprendre qu'une action ait été exécutée en réaction à une provocation, ou suite à un choix délibéré de l'adversaire. Cependant, l'identification d'une action comme étant une réaction à une provocation est un problème en soi.

Un problème majeur de notre algorithme de provocation est l'hypothèse que l'adversaire ne sait pas qu'il se fait provoquer. En effet, nous supposons que l'adversaire réagira à une provocation sans se douter que sa réaction pourrait involontairement révéler son intention, alors qu'il pourrait délibérément réagir de façon ambiguë, ou même provoquer à son tour l'observateur, par exemple dans le cadre d'une tactique de déception. Ce type de modélisation récursive de l'état mental de l'adversaire est directement abordé par la théorie des jeux [56] ; une investigation plus approfondie de ce cadre théorique et de son application à la reconnaissance de plan serait intéressante pour faire suite à ce travail de recherche.

Finalement, le problème de sélection d'une action de provocation dans un domaine avec des adversaires n'est qu'un cas particulier du problème plus général de prise de décision dans un environnement incertain avec des adversaires. En effet, un planificateur explorant toutes les possibilités réussirait (étant donnés des ressources de calcul et un temps infinis) à identifier la meilleure action à prendre, que ce soit une action de provocation ou une action dite régulière. Le problème de prise de décision dans un environnement incertain est abordé par les processus de décision de Markov partiellement observables (POMDP), alors que la généralisation au contexte multiagent est étudiée par les POMDP interactifs (I-POMDP) [40] ou décentralisés (DEC-POMDP) [14]. Puisque les algorithmes de résolution de ces modèles sont très complexes et pratiquement inutilisables dans des applications outre les problèmes très simples (*toy problems*) [41], il serait intéressant d'étudier l'utilisation d'une méthode de sélection d'une provocation comme celle présentée plus tôt pour aider à résoudre un I-POMDP ou un DEC-POMDP de façon heuristique.

Chapitre 4

Expérimentations et résultats

Dans ce chapitre, nous présentons les expérimentations effectuées pour évaluer notre algorithme de reconnaissance de plan ainsi que les résultats obtenus. Notons qu'il est très difficile d'évaluer un algorithme de reconnaissance de plan, en particulier dans notre domaine d'application. En effet, contrairement à d'autres domaines de recherche en intelligence artificielle, comme l'apprentissage automatique et la planification, il n'existe que très peu de jeux de données pour comparer les algorithmes [1] ; il n'existe à notre connaissance aucun ensemble de données pour un domaine avec des adversaires, ni aucun ensemble de données pour tester l'apport de la provocation de l'adversaire.

Pour notre domaine d'application, l'ultime validation d'un algorithme de reconnaissance de plan se ferait en l'intégrant à un algorithme de prise de décision et en analysant les performances (par exemple le nombre de victoires ou le temps moyen avant de gagner) de l'agent intégré par rapport à un autre sans algorithme de reconnaissance de plan. Cependant, une telle évaluation nécessite l'existence d'un processus décisionnel sophistiqué. La prise de décision dans un jeu de stratégie en temps réel étant déjà un problème de recherche extrêmement complexe en soi, il nous était impossible d'implémenter à la fois un algorithme de reconnaissance de plan et un algorithme décisionnel puis ensuite de les intégrer et de comparer les agents. Pour ces raisons, nous avons évalué notre algorithme en mode différé (*offline*), indépendamment du système auquel il serait idéalement intégré.

1. Certains jeux de données sont disponibles à cette adresse : `http://www.planrec.org/`.

Nous commençons donc par présenter les métriques que nous avons utilisées pour évaluer notre algorithme. Nous décrivons ensuite les différentes expérimentations, puis nous analysons les résultats obtenus.

4.1 Métriques

4.1.1 Matrice de confusion

Plusieurs des métriques que nous avons utilisées pour évaluer notre algorithme sur plusieurs scénarios se basent sur une matrice de confusion (tableau 4.1). Nous reprenons la même méthodologie que Ramírez et Geffner [65] pour traduire les résultats probabilistes de la reconnaissance de plan en une classification binaire et ainsi construire la matrice de confusion. Nous commençons donc par calculer l'ensemble G' des buts qui maximisent la probabilité $P(g \mid \mathcal{O})$ [2] parmi tous les buts $g \in \mathcal{G}$ (voir l'équation (3.6)). Si le but réel G maximise cette probabilité *a posteriori*, c'est-à-dire que $G \in G'$, alors il s'agit d'un vrai positif (VP). Le nombre de faux positifs (FP) est égal à $|G'| - 1$ et le nombre de vrais négatifs (VN) est égal à $|\mathcal{G}| - |G'|$. Par contre, si le but réel ne maximise pas la probabilité *a posteriori*, alors il s'agit d'un faux négatif (FN). Le nombre de faux positifs (FP) est alors égal à $|G'|$ et le nombre de vrais négatifs (VN) est égal à $|\mathcal{G}| - |G'| - 1$.

Tableau 4.1 – Matrice de confusion pour le problème de classification binaire

		Classe prédite	
		$G \in G'$	$G \notin G'$
Classe réelle	$G \in G'$	VP	FN
	$G \notin G'$	FP	VN

À partir de cette matrice de confusion, nous pouvons calculer les métriques listées dans le tableau 4.2, couramment utilisées pour évaluer les algorithmes d'apprentissage automatique et de classification [3, 67].

La précision P mesure la proportion des buts les plus probables qui sont correctement identifiés, c'est-à-dire qui correspondent avec la réalité de terrain (*ground*

2. Deux valeurs x et x' sont considérées égales si $|x - x'| < 10^{-7}$.

Tableau 4.2 – Métriques dérivées de la matrice de confusion

Métrique	Symbole	Équation
Précision	P	VP / (VP + FP)
Rappel	R	**VP / (VP + FN)**
Spécificité	S	VN / (VN + FP)
Exactitude	E	**(VP + VN) / (VP + FN + FP + VN)**
Score F_1	F_1	$2 \times (P \times R) / (P + R)$

truth). Le rappel R (aussi appelé la sensibilité) mesure la proportion des bons buts qui sont correctement identifiés comme étant les plus probables. La spécificité S mesure la proportion de mauvais buts qui sont correctement identifiés comme ne faisant pas partie des buts les plus probables. L'exactitude E mesure la proportion de buts qui sont correctement identifiés comme étant les bons ou non parmi tous les buts. Enfin, le score F_1 donne la moyenne harmonique entre la précision et le rappel. La valeur optimale de chacune de ces métriques est 1, alors qu'une valeur de 0 signifie que l'algorithme se trompe complètement.

4.1.2 Coefficient Kappa

À partir de la matrice de confusion, nous pouvons aussi calculer le coefficient Kappa $\kappa = \frac{Pr(a)-Pr(e)}{1-Pr(e)}$, où $Pr(a) = E$ est l'accord relatif entre l'algorithme et la réalité de terrain et où $Pr(e)$ est la probabilité d'un accord aléatoire [31, 50]. Cette métrique est donc considérée plus robuste que l'exactitude seule, puisqu'elle tient aussi compte d'un accord survenant par chance entre l'algorithme et la réalité de terrain. Le tableau 4.3 donne une interprétation des valeurs du coefficient Kappa.

Tableau 4.3 – Degré d'accord selon la valeur du coefficient Kappa [50]

κ	Accord
< 0.00	Désaccord
0.00 — 0.20	**Faible**
0.21 — 0.40	Passable
0.41 — 0.60	**Moyen**
0.61 — 0.80	Considérable
0.81 — 1.00	**Presque parfait**

4.1.3 Point de convergence

Nous utilisons aussi le point de convergence [18, 52] comme métrique de qualité pour l'algorithme de reconnaissance de plan. Cette mesure indique le moment à partir duquel le bon but conserve la probabilité *a posteriori* la plus élevée jusqu'à la fin du scénario. En d'autres termes, le point de convergence donne le délai que met l'algorithme à correctement et définitivement reconnaître le plan de l'agent observé.

Le point de convergence est calculé en pourcentage de complétion d'un scénario (où 0 % correspond au début d'un scénario et où 100 % correspond à sa fin), de façon à accommoder et à pouvoir comparer les scénarios de différentes durées. Plus le point de convergence est bas, plus l'algorithme est capable de comprendre rapidement l'intention de l'agent observé. Un point de convergence de 100 % indique que l'algorithme a nécessité l'observation de toutes les actions avant de pouvoir correctement identifier le plan, ou qu'il n'a tout simplement pas réussi à le faire avant la fin du scénario.

4.1.4 Progression des probabilités dans le temps

Bien que les métriques dérivées de la matrice de confusion soient de bons indicateurs de performance, la construction de cette matrice transforme le problème de reconnaissance de plan probabiliste en un problème de classification binaire. Ainsi, ces mesures ne tiennent pas compte de la distribution de probabilités $P(G \mid \mathcal{O})$ des buts étant données les observations, mais seulement de l'ensemble des buts les plus probables sans vraiment tenir compte de l'écart entre les probabilités conditionnelles (outre le seuil d'égalité mentionné précédemment). Pour cette raison, nous mesurons aussi la justesse de la reconnaissance de plan en mesurant la probabilité du bon but (selon le scénario) et sa progression dans le temps, au fur et à mesure que des actions sont observées.

4.2 Stratégies d'ouverture

Les joueurs professionnels de jeux de stratégie en temps réel développent avec le temps et l'expérience des stratégies d'ouverture (*build orders*). Ces stratégies dictent l'ordre de construction des structures ainsi que la quantité et le type d'unités à recruter

avant de pouvoir lancer la première attaque contre l'adversaire en début de partie.

Dans le jeu *StarCraft : Brood War*, un favori chez les joueurs professionnels, les stratégies d'ouverture sont bien connues et documentées, particulièrement sur le site Wiki de la communauté *Team Liquid*[3]. Ces stratégies diffèrent selon l'affrontement, c'est-à-dire les races choisies par chacun des joueurs. Nous avons sélectionné pour notre expérimentation les stratégies d'ouverture d'un adversaire de race *Zerg* contre un observateur de race *Terran*. La bibliothèque de plans pour ces stratégies est donnée dans la section B.1.

Nous avons extrait 155 scénarios de reprises de parties entre des joueurs professionnels de *StarCraft : Brood War*. Chacun de ces scénarios contient une séquence d'observations de construction de structures et de recrutement d'unités ainsi que le temps auquel ces actions ont été observées ; les plans ont été étiquetés à la main, un travail long et laborieux qui explique la quantité relativement faible de scénarios utilisés pour l'expérimentation. Le tableau 4.4 détaille le nombre de scénarios où l'adversaire suit chacun des plans[4] ainsi que la probabilité *a priori* de chacun des plans, plus ou moins égale à la proportion de scénarios de ce type.

Tableau 4.4 – Stratégies d'ouverture pour le jeu *StarCraft : Brood War*

Nom du plan	Nombre de scénarios	Probabilité *a priori*
4 Pool	2	0.05
5 Pool	2	0.05
9 Pool	34	0.2
12 Pool	13	0.1
12 Hatch	104	0.6

Le tableau 4.5 indique le temps moyen et le nombre moyen d'actions pour compléter chacun des plans. Le nombre d'actions pour compléter les scénarios est supérieur au nombre d'actions dans les plans (voir la bibliothèque de plans dans la section B.1) parce que certaines actions superflues sont ignorées par l'algorithme de reconnaissance de plan[5].

3. `http://wiki.teamliquid.net/starcraft/`

4. Les stratégies d'ouverture sont bien entendu mutuellement exclusives.

5. L'algorithme de reconnaissance de plan ignore les observations qui ne peuvent être ajoutées à aucune hypothèse existante et qui ne génèrent donc aucune nouvelle hypothèse.

Tableau 4.5 – Temps et nombre d'actions en moyenne pour compléter les scénarios

Nom du plan	Temps (s)	Nombre d'actions
4 Pool	93.5	6.0
5 Pool	**97.5**	**10.0**
9 Pool	121.65	22.62
12 Pool	**160.0**	**31.77**
12 Hatch	177.36	34.72

La figure 4.1 illustre la progression moyenne de la probabilité de chacun des buts en fonction de la complétion des scénarios. Nous remarquons tout d'abord que tous les plans finissent par être reconnus correctement par l'algorithme avec une probabilité de 1 ; nous avons donc omis de rapporter les valeurs des métriques dérivées de la matrice de confusion, puisqu'elles auraient toutes une valeur parfaite de 1. Nous pouvons aussi y observer que les plans *4 Pool* et *5 Pool* sont rapidement identifiés par l'algorithme de reconnaissance de plan malgré leur faible probabilité *a priori*. Ces stratégies, où l'adversaire mise le tout pour le tout dans la première attaque avec laquelle il espère surprendre son rival, sont brèves (moins de 100 s) ; l'algorithme de reconnaissance de plan est capable d'éliminer les hypothèses portant sur les autres stratégies de plus longues durées pour reconnaître le bon plan. Le plan *9 Pool* exhibe aussi une bonne courbe de progression de la précision, monotone croissante sur presque toute la durée de l'exécution de ce plan.

L'algorithme met plus de temps pour reconnaître correctement les plans *12 Pool* et *12 Hatch*, plus longs et assez semblables l'un et l'autre. Le creux dans les courbes de précision de ces plans est dû à la façon dont l'algorithme calcule les probabilités des hypothèses explicatives : plus il y a d'observations appartenant à un plan, plus la probabilité de ce plan (en particulier le facteur $P(\mathcal{O} \mid \eta)$, voir l'équation (3.4)) sera élevée. Puisque le début des stratégies *9 Pool*, *12 Pool* et *12 Hatch* est à peu près identique (voir la section B.1), l'algorithme accorde une plus grande probabilité au plan *9 Pool* parce qu'il ne reste que très peu d'actions à exécuter pour compléter ce plan. Dès que cette hypothèse est infirmée par l'observation d'actions non compatibles avec le plan *9 Pool*, les probabilités des plans *12 Pool* et *12 Hatch* remontent graduellement jusqu'à ce que ces deux plans soient correctement distingués.

Cette explication est confirmée par la figure 4.2 qui illustre la progression moyenne

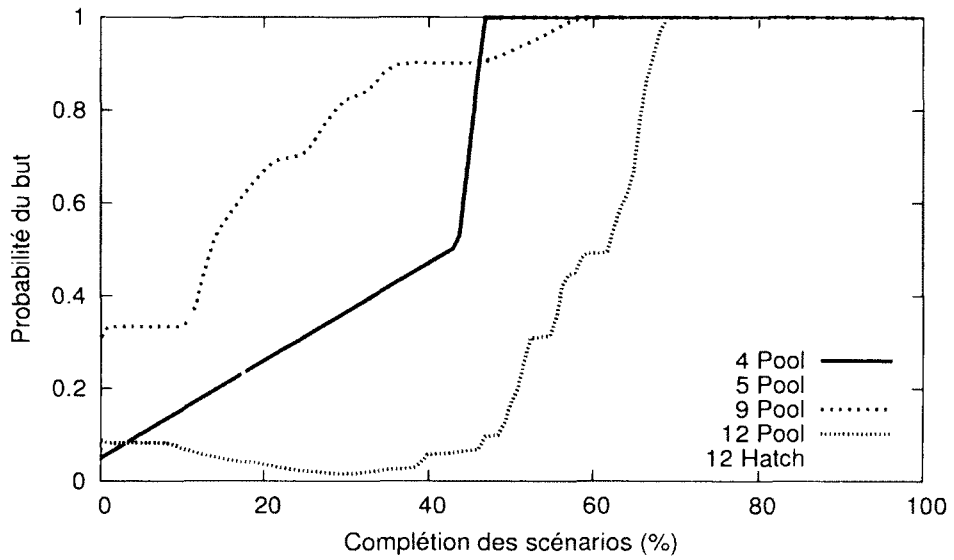


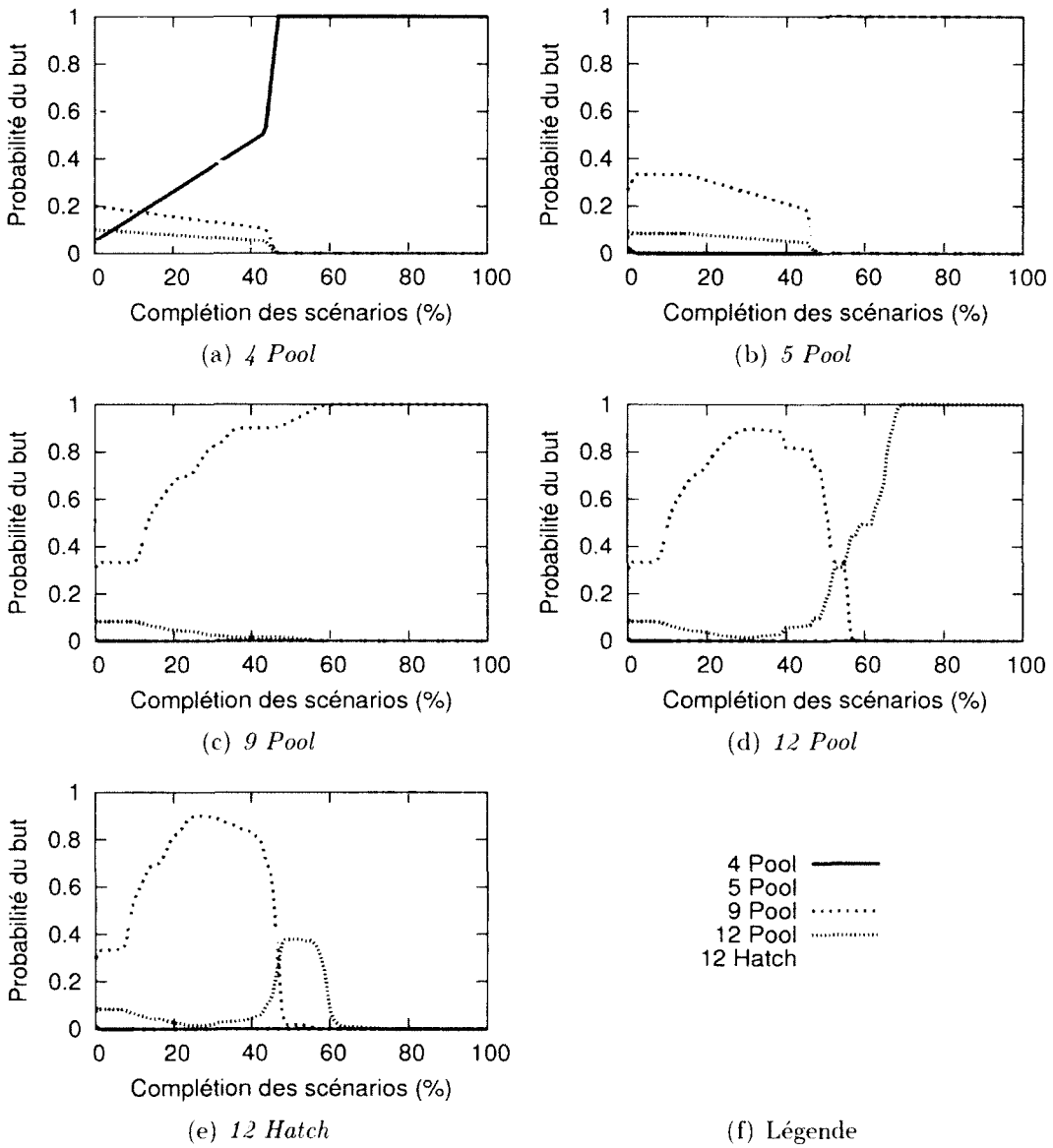
Figure 4.1 – Progression moyenne des probabilités en fonction de la complétion du plan

des probabilités conditionnelles de chacun des plans étant données les observations pour chacun des types de scénarios. En particulier, les figures 4.2(d) et 4.2(e) montrent la courbe du plan *9 Pool* qui augmente jusqu'à ce que ce plan soit invalidé, après quoi les probabilités des plans *12 Pool* et *12 Hatch* commencent à monter.

Enfin, nous avons mesuré le point de convergence moyen de chaque plan dans les différents scénarios. Les résultats, donnés dans le tableau 4.6, indiquent que c'est le plan *9 Pool* qui converge le plus rapidement, pour la même raison qu'auparavant. Le plan *12 Pool* est celui qui converge le plus lentement puisqu'il est très semblable au plan *12 Hatch*, mais a une probabilité *a priori* bien inférieure à celle de ce dernier.

Tableau 4.6 – Point de convergence moyen selon la stratégie d'ouverture

Stratégie	Point de convergence (%)
4 Pool	41.667
5 Pool	**44.949**
9 Pool	17.761
12 Pool	**68.976**
12 Hatch	45.363



(a) *4 Pool* (b) *5 Pool*

(c) *9 Pool* (d) *12 Pool*

(e) *12 Hatch* (f) Légende

Figure 4.2 – Progression moyenne des probabilités selon les scénarios

4.3 Scénario typique d'un jeu RTS

La reconnaissance des stratégies d'ouverture demeure cependant une application simpliste de la reconnaissance de plan. En effet, le nombre d'hypothèses est toujours limité au nombre de plans dans la bibliothèque à cause d'une action placée au début de chacun de ces plans marquant le commencement du scénario. Cette action artificielle s'assure que l'algorithme ne considère pas la possibilité que l'agent puisse suivre plusieurs stratégies d'ouverture simultanément, sans avoir à modifier le fonctionnement de l'algorithme. Cependant, chacun des joueurs dans un jeu de stratégie en temps réel exécutera simultanément plusieurs plans en général, soit pour atteindre plusieurs objectifs ou encore pour contrôler différents groupes d'unités de façon concurrente.

Pour cette raison, nous avons choisi un scénario plus complexe qui s'inspire d'une partie typique d'un jeu de stratégie en temps réel (section A.12) pour l'évaluation de notre algorithme. Le scénario oppose à l'observateur un agent commandant à 2 groupes d'unités d'exécuter une séquence de 25 actions au total pour atteindre 5 objectifs durant le déroulement du scénario. Les actions faisant partie des plans suivis par chaque groupe d'unités (2 plans pour le 1er groupe et 3 plans pour le 2^{e} groupe) sont exécutées séquentiellement. Le scénario pourrait donc être divisé en 2 sous-scénarios (1 par groupe d'unités) ou même en 5 sous-scénarios (1 par objectif) indépendants. Nous avons utilisé la bibliothèque de comportements tactiques décrite dans la section B.3 et chacun des plans a une probabilité *a priori* égale à 0.2.

Nous avons donc mesuré la progression dans le temps de la probabilité du bon but étant données les observations. Nous considérons que le but est la combinaison des $1 \leq n \leq 5$ plans pour lesquels l'agent a commencé à exécuter des actions, c'est-à-dire que le but ne contient qu'un seul plan tant que des actions appartenant au 2^{e} n'ont pas encore été exécutées, et ainsi de suite[6]. Nous avons comparé notre algorithme (PROBE) avec Yappr[7] [38] (*Yet Another Probabilistic Plan Recognizer*), un algorithme très semblable à PHATT, mais qui utilise une grammaire PFFG (*plan frontier fragment grammar*) comme représentation interne des plans plutôt que des arbres « ET/OU »

6. Si vous avions considéré dès le départ que le but de l'agent était la combinaison des 5 plans à exécuter durant le scénario, la probabilité *a posteriori* aurait été égale à 0 jusqu'à ce que des actions appartenant au 5^{e} plan (soit à 80 % de la complétion du scénario) commencent à être observées.

7. Le code de Yappr a gracieusement été fourni par John Maraist, un des auteurs de l'algorithme.

similaires aux réseaux de tâches hiérarchiques. Cette conversion automatisée grâce à un prétraitement sur la bibliothèque de plans permet de réduire la complexité de l'algorithme de génération des hypothèses et de calcul des probabilités ainsi que d'alléger l'utilisation de la mémoire vive. Les résultats de la comparaison sont illustrés dans la figure 4.3.

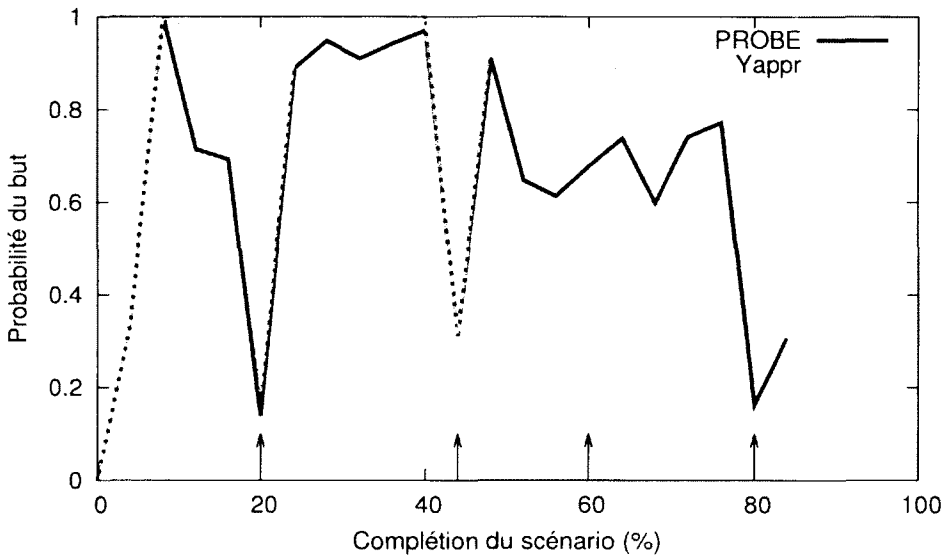


Figure 4.3 – Progression des probabilités en fonction de la complétion du scénario

Tout d'abord, la raison pour laquelle les courbes s'arrêtent avant la fin du scénario est que chacun des algorithmes a fini par manquer de mémoire vive (un total de 1 Go de mémoire vive était alloué à chaque algorithme) pour générer des hypothèses : 21 actions ont pu être observées avant que notre algorithme ne manque de mémoire et 16 actions ont pu être observées avant que Yappr ne s'arrête. Les flèches au bas de la figure indiquent les moments où l'agent observé a commencé à exécuter des actions appartenant à un nouveau plan, et donc au moment où le but de l'agent a été modifié pour mesurer la probabilité conditionnelle étant données les observations.

Une première lecture des courbes semble indiquer que Yappr est meilleur que notre algorithme pour reconnaître l'intention de l'agent dans ce scénario. Cependant, nous avons aussi calculé la valeur moyenne des différentes métriques dérivées de la matrice de confusion à différents moments durant le scénario. Les résultats obtenus (rapportés

dans le tableau 4.7) sont exactement les mêmes pour les deux algorithmes (outre le fait que Yappr se soit arrêté avant notre algorithme dû à un manque de mémoire). Cela révèle donc que notre algorithme réussit aussi bien que Yappr à discriminer le bon but de l'agent observé parmi tous les buts possibles, malgré que les probabilités *a posteriori* soient parfois inférieures à celles calculées par Yappr.

Tableau 4.7 – Qualité moyenne de la reconnaissance de plan avec PROBE et Yappr

$\lvert\mathcal{O}\rvert$	P	R	S	E	F_1	κ
5	0.571	0.800	0.750	0.765	0.667	0.493
10	0.750	0.900	0.923	0.918	0.818	0.766
15	0.737	0.933	0.975	0.972	0.824	0.809
20	0.720	0.900	0.991	0.989	0.800	0.794

La différence entre les probabilités calculées par les algorithmes semble donc avoir relativement peu d'importance sur la qualité de la reconnaissance de plan ; un ajustement des probabilités *a priori* des plans dans la bibliothèque permettrait par exemple de réduire l'écart entre les probabilités. La raison expliquant la différence est sans doute la représentation des plans de la bibliothèque qui diffère pour chaque algorithme : PROBE utilise des machines à états finis alors que Yappr utilise des arbres « ET/OU » transformés en grammaires PFFG. En effet, même si les deux représentations produisent les mêmes séquences d'actions, le nombre d'hypothèses générées suite à l'observation de ces actions diffère grandement, comme le démontrent les valeurs dans le tableau 4.8. Le tableau donne aussi le temps requis pour générer l'ensemble des hypothèses ainsi que pour calculer la probabilité $P(G \mid \mathcal{O})$ du bon but G ; notons cependant que notre algorithme est implémenté en Java alors que Yappr est implémenté en Common Lisp.

La différence entre les quantités d'hypothèses générées est causée par la représentation d'arbres « ET/OU » (ou des grammaires PFFG) : chaque état d'une machine à états finis ayant plus d'une transition nécessite un nœud « OU », et Yappr (tout comme PHATT) génère une hypothèse différente pour chaque choix d'enfant sous un nœud « OU ». Quant au grand nombre d'hypothèses générées dans les deux cas, cela est causé par l'algorithme itératif qui génère toutes les hypothèses valides avec la séquence d'observations (voir l'algorithme 3.1 et les figures 3.5 et 3.6). Prenons par exemple le comportement de déplacement défensif de la figure B.3(a) et imaginons la

Tableau 4.8 – Nombre d'hypothèses générées durant le scénario

$\lvert\mathcal{O}\rvert$	Algorithme	Hypothèses	Génération (s)	Calcul de $P(G \mid \mathcal{O})$ (s)
5	PROBE	5	0.001	0.002
	Yappr	44	0.013	0.001
10	**PROBE**	**8**	**0.001**	**0.008**
	Yappr	**432**	**0.091**	**0.003**
15	PROBE	248	0.021	0.031
	Yappr	129 600	34.807	2.065
20	**PROBE**	**21 360**	**0.595**	**3.986**
	Yappr	—	—	—
21	PROBE	75 360	2.595	15.255
	Yappr	—	—	—

séquence d'observations suivante :

$$\mathcal{O} = \langle MoveTo(\mathsf{U1_E}, \mathsf{S1_A}), MoveAway(\mathsf{U1_E}, \mathsf{U1_A}), MoveTo(\mathsf{U1_E}, \mathsf{S1_A}) \rangle$$

L'observation de la dernière action produira au moins deux hypothèses : une expliquant cette action comme faisant partie de la même instance du comportement de déplacement offensif, et une autre expliquant cette action comme débutant une nouvelle instance du même comportement. Si la bibliothèque de plans contient d'autres plans débutant par la même action (comme c'est notre cas, voir la figure B.3), le nombre d'hypothèses générées augmente en conséquence.

Cette particularité de l'algorithme PHATT, sur lequel se basent PROBE ainsi que Yappr, est un frein pour la reconnaissance de comportements cycliques qui finit par causer des problèmes d'utilisation de la mémoire, comme nous avons déjà pu le constater. Un algorithme comme ELEXIR [37] ne souffrant pas de ce problème n'aurait sans doute pas autant de difficulté à reconnaître l'intention de l'agent dans le scénario présenté dans cette section. Malgré tout, le formalisme de machines à états finis que nous avons choisi pour les plans dans notre algorithme semble être mieux adapté que les grammaires PFFG ou les arbres « ET/OU » pour des comportements cycliques, puisqu'ils font en sorte que l'algorithme génère moins d'hypothèses : l'utilisation de la mémoire vive est donc quelque peu allégée. De plus, la représentation graphique des machines à états finis est, selon nous, plus intuitive qu'avec les autres formalismes,

facilitant ainsi la conception d'une bibliothèque de plans pour un domaine donné.

Il convient cependant de rappeler que les machines à états finis sont moins expressives que les arbres « ET/OU ». Cela nous a par contre permis d'optimiser l'implémentation de l'algorithme, comme les temps d'exécution dans le tableau 4.8 le montrent. De plus, afin de réduire au maximum l'utilisation de la mémoire, notre algorithme conserve le moins d'informations possible pour générer les hypothèses : quand les probabilités doivent être calculées, les informations manquantes sont d'abord récupérées, ce qui occasionne une complexité supplémentaire pour le calcul des probabilités qui se reflète dans le temps de calcul de la probabilité $P(G \mid \mathcal{O})$. Ainsi, les avantages de Yappr par rapport à PHATT (qui doit constamment copier des arbres « ET/OU » partiellement instanciés [38]) se font très peu sentir dans une comparaison avec notre approche. Le choix entre les deux algorithmes se réduit donc à un choix ou à une préférence quant au formalisme de représentation des plans de la bibliothèque.

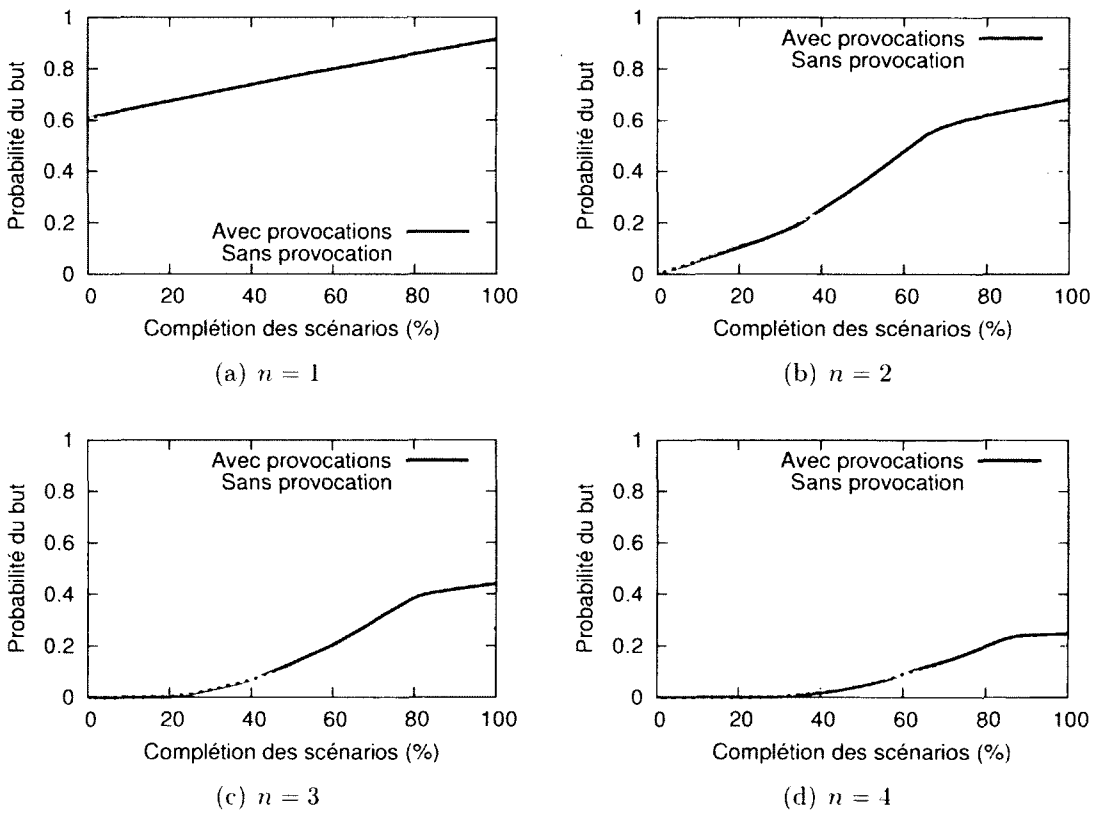
4.4 Provocation de l'adversaire

Afin de valider l'algorithme de provocation de l'adversaire, nous avons généré aléatoirement des scénarios. Pour ce faire, nous avons utilisé la même bibliothèque de comportements que dans la section 4.3 (voir la figure B.3) ; nous avons encore une fois donné une probabilité *a priori* de 0.2 à chacun de ces comportements. Ces plans sont relativement similaires les uns aux autres, donc difficiles à reconnaître précisément puisque leur exécution génère des séquences d'observations semblables, parfois ayant même plusieurs explications valides. Notre hypothèse est que la provocation de l'agent observé permet d'améliorer la précision et la rapidité de la reconnaissance des intentions de celui-ci.

La figure 4.4 montre la progression moyenne de la probabilité du bon but en fonction de la complétion des scénarios. Ces scénarios ont été générés en sélectionnant aléatoirement n plans de la bibliothèque, puis en laissant l'agent observé exécuter des actions appartenant à ces plans. Les seules actions permises sont celles suivant des événements ne pouvant pas être provoqués. Chacune des courbes représente la moyenne sur 500 scénarios. Contrairement à la section 4.3, nous considérons ici dès le début des scénarios que le but de l'agent est l'ensemble des n plans choisis aléa-

toirement, puisque ces plans sont exécutés aléatoirement de façon entrelacée, et non séquentiellement : il aurait donc été difficile de calculer des moyennes tout en faisant évoluer le but de l'agent pendant chacun des 500 scénarios.

Nous avons donc comparé la situation où l'observateur peut provoquer l'agent observé à celle où il ne peut pas le provoquer afin de mesurer l'apport de la provocation de l'agent observé par rapport à une méthode entièrement passive. Lorsque l'observateur provoque l'agent observé, ce dernier sélectionne aléatoirement une action à exécuter parmi les réactions possibles à la provocation. Dans chaque cas, nous avons utilisé les valeurs $\gamma = 0.5$ et $\delta = 0.5$ pour les paramètres de l'algorithme de provocation.



(a) $n = 1$ (b) $n = 2$

(c) $n = 3$ (d) $n = 4$

Figure 4.4 - Progression moyenne des probabilités selon le nombre n de plans

Les courbes ont toutes des allures semblables, mais elles sont plus ou moins éti-

rées et inclinées selon le nombre de plans dans les scénarios. Dans tous les cas, la provocation améliore la justesse finale de l'algorithme de reconnaissance de plan : dans les scénarios avec $n = 2$ plans par exemple, la provocation a permis d'augmenter la probabilité que le bon plan soit reconnu correctement à la fin du scénario de 71.6 %, faisant passer la probabilité de 0.397 à 0.682. Enfin, les distances entre les courbes montrent que la provocation permet en moyenne d'améliorer la qualité de la reconnaissance, et ce plus tôt dans le déroulement des scénarios.

Les métriques de qualités résumées dans le tableau 4.9 et les points de convergence moyens, donnés dans le tableau 4.10, confirment cette affirmation. Ces valeurs donnent la moyenne des résultats obtenus à la fin des 500 scénarios générés aléatoirement. Notons cependant que la qualité de la reconnaissance est réduite assez considérablement lorsque le nombre de plans constituant le but de l'agent dans les différents scénarios augmente, en particulier parce que la probabilité *a posteriori* est égale à 0 jusqu'à ce qu'au moins une action appartenant à chacun des plans composant le scénario ait été observée.

Tableau 4.9 – Qualité de la reconnaissance selon le nombre n de plans

	Provocation(s)	P	R	S	E	F_1	κ
$n = 1$	Oui	1.000	1.000	1.000	1.000	1.000	1.000
	Non	0.620	0.620	0.676	0.650	0.620	0.296
$n = 2$	Oui	0.777	0.780	0.931	0.895	0.778	0.710
	Non	0.389	0.540	0.699	0.658	0.452	0.213
$n = 3$	Oui	0.513	0.536	0.914	0.859	0.524	0.442
	Non	0.318	0.394	0.832	0.759	0.352	0.206
$n = 4$	Oui	0.318	0.340	0.928	0.875	0.329	0.260
	Non	0.323	0.346	0.917	0.858	0.334	0.255

Tableau 4.10 – Point de convergence moyen selon le nombre n de plans

	Avec provocations (%)	Sans provovocation (%)
$n = 1$	16.067	28.700
$n = 2$	55.270	69.879
$n = 3$	77.580	89.807
$n = 4$	89.259	96.573

Nous remarquons aussi certaines situations (en particulier dans la figure 4.4(b)

entre environ 0 % et 35 % de la complétion des scénarios) où la provocation empire légèrement la situation. Ces situations surviennent lorsque les probabilités de mauvaises hypothèses augmentent suite à une provocation, réduisant du même coup l'incertitude sur les intentions de l'adversaire (ce qui explique pourquoi la provocation a effectivement eu lieu). Cela indique que l'algorithme de provocation, dans son état actuel, dépend fortement de la justesse de l'algorithme de reconnaissance de plan sous-jacent.

Nous avons ensuite étudié les effets de la variation des paramètres γ et δ sur le processus de reconnaissance de plan. Pour ce faire, nous avons fixé le nombre de plans constituant le but de l'agent à $n = 2$ et nous avons mesuré la progression des probabilités du but en fonction de la complétion des scénarios, comme précédemment, mais avec différentes valeurs pour les paramètres. La figure 4.5 illustre cette progression des probabilités et le tableau 4.11 identifie la courbe associée à chaque combinaison des valeurs des paramètres γ et δ. Nous avons aussi rapporté les résultats pour une stratégie de provocation complètement aléatoire (le choix de provoquer ou non ainsi que le choix de l'événement à provoquer, le cas échéant) afin de vérifier si notre approche basée sur la mesure d'entropie produit de meilleurs résultats. La courbe J correspond à cette stratégie de sélection de provocations aléatoire.

Tableau 4.11 – Légende de la figure 4.5

	$\gamma = 0.0$	$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 0.75$	$\gamma = 1.0$
$\delta = 0.0$		E	F	G	
$\delta = 0.25$	B	B	B	B	
$\delta = 0.5$	C	C	C	C	
$\delta = 0.75$					
$\delta = 1.0$	D	D	D	D	D

Les résultats indiquent que le paramètre γ, spécifiant l'entropie minimale pour rechercher une provocation, a la plus grande influence lorsque la valeur du paramètre δ, spécifiant la désambiguïsation minimale pour provoquer, est basse. Il s'agit d'un effet à prévoir, puisque le paramètre γ devient alors la seule base sur laquelle repose la décision de provoquer ou non l'adversaire. Comme la figure 4.5 le montre, une combinaison de valeurs trop faibles pour les paramètres γ et δ produit les résultats les moins bons au début du scénario, bien que la situation finisse par s'améliorer avec

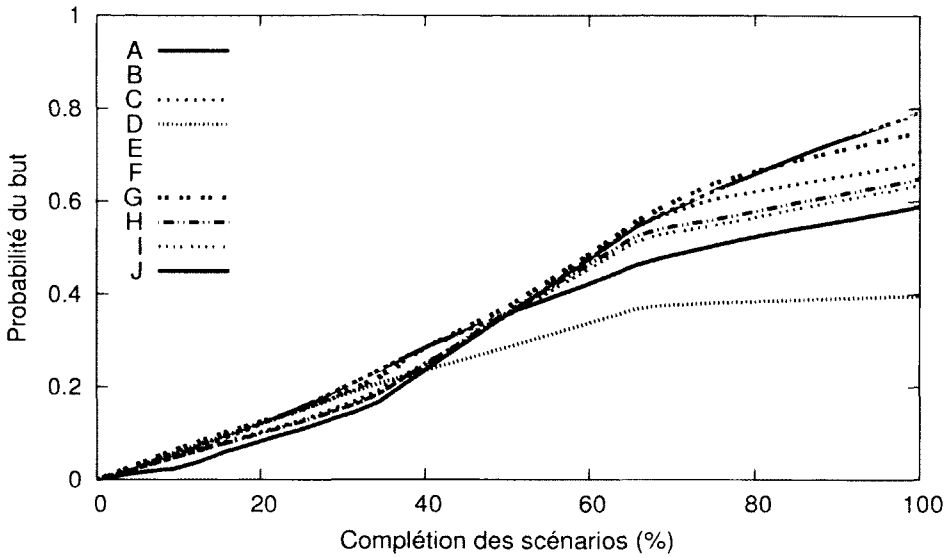


Figure 4.5 – Effets de la variation des paramètres sur les scénarios avec $n = 2$ plans

le temps (courbe A). Cela démontre qu'être trop enclin à provoquer l'adversaire peut engendrer une baisse de la qualité de la reconnaissance des intentions de ce dernier. À l'inverse, une valeur de γ trop stricte produit une réduction de la justesse de la reconnaissance à long terme (courbes H et I).

Le paramètre δ a une plus grande influence sur la reconnaissance de plan. Les résultats semblent indiquer que la précision est meilleure plus la valeur de δ diminue, mais que le paramètre γ doit être judicieusement choisi. Encore une fois, une valeur trop stricte pour ce paramètre réduit grandement la qualité de la reconnaissance : la courbe D correspond aussi à la situation où l'adversaire n'est jamais provoqué.

Nous croyons cependant que le choix des bonnes valeurs pour les paramètres γ et δ repose fortement sur le domaine et sur la structure de la bibliothèque de plans utilisée ; l'influence de ces paramètres sur la qualité de la reconnaissance de plan est complexe et mériterait d'être étudiée plus profondément, par exemple sur d'autres domaines ou encore avec des bibliothèques de plans différentes.

Comme nous l'avons évoqué au chapitre 3, nous pourrions imaginer des situations où provoquer l'adversaire empire la situation malgré la réduction espérée de l'ambiguïté. En effet, si la réaction de l'adversaire suite à une provocation est compatible

avec deux hypothèses et que la mauvaise devient privilégiée par rapport à la bonne (soit à cause d'une probabilité *a priori* plus élevée ou à cause de la structure des plans), l'ambiguïté sur les intentions de l'adversaire sera réduite. Dans ces situations, s'abstenir de provoquer (grâce à des seuils γ et δ plus élevés) garde la probabilité de la bonne hypothèse plus haute, au prix d'une incertitude plus grande.

Il serait peut-être aussi nécessaire d'utiliser d'autres facteurs qu'uniquement la réduction espérée de l'ambiguïté sur les intentions de l'adversaire, comme par exemple dans l'équation (3.8), puisque cette seule mesure ne semble pas toujours être suffisante pour sélectionner la meilleure provocation. Néanmoins, les résultats démontrent que provoquer l'adversaire aide à améliorer la précision ainsi que la rapidité de détection de l'algorithme de reconnaissance de plan. De plus, même une stratégie aléatoire de provocation demeure généralement une meilleure solution que de s'abstenir complètement de provoquer l'adversaire, mais est tout de même inférieure à notre algorithme de sélection d'une provocation basée sur une mesure d'entropie.

Chapitre 5

Travaux futurs

Dans ce chapitre, nous proposons quelques perspectives pour l'amélioration de certains aspects des travaux réalisés dans le cadre de ce travail de recherche. Nous décrivons donc brièvement sept axes à explorer pour généraliser notre algorithme.

5.1 Interactions entre observateur et observé

Comme nous l'avons mentionné à la fin du chapitre 3, notre algorithme de provocation (de même que l'algorithme de reconnaissance de plan) ne tient pas compte des interactions entre l'observateur et l'agent observé. En effet, nous supposons que l'adversaire réagira à une provocation sans se douter que sa réaction pourrait involontairement révéler son intention. Celui-ci pourrait délibérément réagir de façon ambiguë, ou même provoquer à son tour l'observateur, par exemple dans le cadre d'une tactique de déception. L'adversaire adaptera ses plans en temps réel en fonction de sa reconnaissance des intentions de l'observateur pour exploiter les faiblesses de ce dernier et *vice versa*.

Cette modélisation récursive de l'état mental de l'autre est abordée par la théorie des jeux [56], en particulier les processus de prise de décision de Markov partiellement observables interactifs (I-POMDP) [40]. L'approche de Pynadath et Wellman [61] présentée au chapitre 2 permet aussi de modéliser les croyances de l'observateur sur l'état mental de l'agent observé, mais pas récursivement. Ces cadres théoriques permettent de modéliser les interactions entre les deux agents et de mieux choisir

comment provoquer l'adversaire, compte tenu de ses réactions possibles et de son état mental, en particulier sa propre représentation de l'état mental de l'observateur, et ainsi de suite.

5.2 Observabilité partielle

Il est généralement irréaliste de supposer que toutes les actions de l'adversaire seront observées. En effet, dans un jeu de stratégie en temps réel, la visibilité de la carte de jeu dépend des unités et des structures d'un joueur, chacune ayant son propre rayon de visibilité. Dans la guerre navale, la capacité d'une force opérationnelle à observer les actions des entités dans l'environnement est limitée par les différents capteurs disponibles. Dans les deux cas, les adversaires tentent de masquer leurs intentions et font tous les efforts nécessaires pour nuire à la reconnaissance de leurs intentions.

Geib et Goldman [35] ont proposé une extension à leur algorithme PHATT permettant de tenir compte des actions exécutées mais non observées. Cependant, cette extension impose une limite sur le nombre d'observations manquées à cause de la façon dont l'algorithme génère les hypothèses : l'absence d'une limite produirait un nombre infini d'hypothèses. De plus, l'ajout de l'observabilité partielle vient complexifier encore davantage un algorithme déjà complexe [33]. L'investigation d'une autre approche capable à la base de gérer l'observabilité partielle s'avérerait peut-être nécessaire pour obtenir un algorithme utilisable dans une application réelle.

5.3 Comportements trompeurs

Les tactiques visant à induire l'ennemi en erreur et à le surprendre avec une attaque sont très souvent utilisées dans les jeux de stratégie en temps réel ainsi que dans les affrontements militaires. Comme l'ont indiqué Elsaesser et Stech [29], la déception fonctionne parce qu'elle exploite les erreurs de raisonnement, les limitations cognitives et les biais concomitants. Bien que la provocation de l'adversaire puisse parfois aider à reconnaître une tactique de déception, un algorithme dédié à la détection de ces tactiques (comme la méthode de Elsaesser et Stech) serait sans doute plus approprié.

5.4 Contraintes de temps réel

Bien que nous ayons testé notre algorithme en mode différé (*offline*), celui-ci est capable de reconnaître les plans de l'adversaire en direct (*online*). Par contre, une caractéristique essentielle d'un algorithme de reconnaissance de plan dans ce contexte est sa capacité à donner une réponse rapidement. En effet, puisque les plans des adversaires changent continuellement en réaction aux actions de l'observateur et aux changements dans l'environnement (voir le chapitre 1), la reconnaissance de plan doit se faire suffisamment vite pour que les résultats ne deviennent pas obsolètes.

En ce sens, il faudrait soumettre l'algorithme de reconnaissance de plan à des contraintes de temps réel afin de s'assurer qu'il puisse fournir une réponse dans un délai raisonnable. Une solution potentielle serait de calculer très rapidement une estimation grossière de l'intention de l'adversaire, puis de raffiner cette estimation s'il reste du temps de calcul. Un algorithme ayant cette caractéristique est dit *anytime* [67]. Une autre solution envisagée serait d'intégrer des heuristiques pour élaguer certaines hypothèses et ainsi empêcher l'explosion du nombre d'hypothèses [79], ou encore des règles analogues aux connaissances pour le contrôle de la recherche en planification automatique en intelligence artificielle [8].

5.5 Comparaison avec d'autres approches

Il serait intéressant de comparer notre algorithme avec d'autres approches de reconnaissance de plan. Par exemple, une implémentation de l'algorithme par planification inverse de Ramírez et Geffner [63, 64, 65] est fournie par les auteurs [1]. Nous avions d'ailleurs tenté de comparer notre algorithme à celui-ci, mais la traduction d'une bibliothèque de plans modélisés en machines à états finis (ou même en arbres « ET/OU ») en un domaine de planification PDDL (*Planning Domain Definition Language*) [48] est une tâche non triviale. Qui plus est, même si le langage PDDL est globalement accepté par les chercheurs en planification en intelligence artificielle, la plupart des algorithmes ne sont compatibles qu'avec un sous-ensemble des fonctionnalités offertes par le langage. Néanmoins, une fois le domaine de planification écrit et

1. https://sites.google.com/site/prasplanning/

un planificateur compatible et efficace trouvé, la comparaison avec notre algorithme devrait pouvoir se faire assez aisément, même si notre approche est non générative. Enfin, d'autres auteurs seraient sans doute disposés à fournir leur algorithme à des fins de comparaison.

5.6 Concept de but mieux défini

Un problème de notre algorithme est la notion de but qui y est quelque peu primitive. En effet, nous considérons que le but de l'agent est simplement d'exécuter ou de compléter l'exécution d'un ou de plusieurs plans de la bibliothèque. Nous pourrions par exemple généraliser le concept de but au sein de notre algorithme en ajoutant des variables et des propositions logiques sur ces variables, modifiées par l'exécution d'actions dans nos machines à états finis ; les buts deviendraient alors des conjonctions et des disjonctions de ces propositions logiques.

Notons que le concept de but dans les approches génératives, ou les approches par planification inverse, n'ont pas ce problème, puisque cette notion est très bien définie dans les planificateurs sous-jacents à ces algorithmes. Une investigation approfondie des approches de cette famille semble donc être une voie très intéressante à suivre pour faire suite à ce projet de recherche.

5.7 Génération de données étiquetées

Comme nous l'avons noté au chapitre 4, il n'existe que très peu d'ensembles de données pour la reconnaissance de plan, contrairement à d'autres domaines d'intelligence artificielle comme la planification et l'apprentissage automatique. De plus, il n'existe à notre connaissance aucun ensemble de données spécifique à la reconnaissance de plan des adversaires. La conception d'un ensemble de données étiquetées pour ce type de problèmes de reconnaissance de plan permettrait donc aux différents chercheurs d'avoir une base comparative pour procéder à l'évaluation leurs algorithmes.

Conclusion

La reconnaissance de plan, soit la compréhension des actions exécutées par une autre personne, est un problème fondamental dans les processus cognitifs humains, mais aussi dans la conception d'agents intelligents. En effet, dans bien des domaines d'application, les agents ont besoin de comprendre les plans de ceux avec qui ils interagissent, directement ou indirectement, ou encore de prédire leurs actions futures.

Les travaux présentés dans ce mémoire de maîtrise s'inscrivent dans le contexte bien précis de reconnaissance de plan des adversaires. L'objectif de ce projet de recherche était de concevoir un algorithme capable de comprendre les intentions d'un agent qui tente de nuire à ce processus. La motivation initiale était la réalisation d'un système d'aide à la décision pour les opérations tactiques navales, dans lequel un algorithme de reconnaissance de plan permettrait d'évaluer plus précisément le niveau de menace associé aux entités observées dans l'environnement. Cependant, nous avons décidé, pour des raisons pratiques, de nous concentrer sur le domaine des jeux de stratégie en temps réel, qui présente des problématiques et des défis très semblables à l'évaluation de la menace dans la guerre navale.

Nous avons donc conçu un algorithme de reconnaissance de plan, fortement inspiré de l'algorithme PHATT, capable de reconnaître en direct les intentions d'un adversaire. L'algorithme est basé sur un modèle d'exécution des plans, ce qui signifie que le processus de reconnaissance de plan est vu comme un mécanisme évoluant dans le temps, au fur et à mesure que les actions constituant ces plans sont observées.

Un problème commun à toutes les applications de la reconnaissance de plan, exacerbé dans les domaines avec des adversaires, est que l'observateur est souvent confronté à des situations où les intentions de l'agent observé sont ambiguës. Afin de pallier ce problème, nous avons conçu un algorithme permettant de sélectionner le

meilleur moyen de provoquer l'agent observé, en espérant que sa réaction donne des indices quant à sa véritable intention. La provocation est donc choisie en fonction de la réduction espérée de l'entropie (comme mesure d'incertitude) associée à l'intention de l'agent observé.

Des expérimentations ont démontré que notre algorithme de reconnaissance de plan réussissait généralement à obtenir une bonne estimation des intentions de l'adversaire dès le départ, et que cette estimation s'améliorait rapidement après chaque nouvelle observation. Nous avons aussi validé l'apport de notre algorithme de provocation et les résultats ont indiqué que cette approche permettait de réduire l'ambiguïté sur les intentions de l'adversaire en sélectionnant une provocation qui le force en quelque sorte à révéler son intention. Les résultats ont aussi montré que la provocation permettait d'améliorer la qualité de la reconnaissance de plan, et que notre stratégie de sélection d'une provocation était généralement meilleure qu'un choix complètement aléatoire de provoquer ou non l'adversaire ainsi qu'un choix aléatoire de la provocation, le cas échéant.

Nous avons aussi comparé notre algorithme avec Yappr, un algorithme inspiré de PHATT qui traduit à l'interne les arbres « ET/OU » de la bibliothèque de plans en grammaires PFFG. Cette transformation permet de régler certains problèmes de PHATT comme l'inefficacité inhérente à la copie d'arbres « ET/OU » partiellement instanciés dans chaque hypothèse et donc l'utilisation de la mémoire vive. La comparaison a cependant démontré que les avantages de Yappr par rapport à PHATT se font beaucoup moins sentir face à notre algorithme, en particulier à cause du formalisme de machines à états finis que nous avons choisi pour notre algorithme, mais aussi grâce à une implémentation efficace qui tente de minimiser l'utilisation de la mémoire vive et qui évite certains calculs complexes jusqu'à ce qu'ils deviennent nécessaires.

Notre algorithme souffre cependant de limitations qui restreignent son applicabilité dans un domaine réel. Nous avons donc suggéré sept avenues possibles pour améliorer et généraliser notre approche, à savoir la prise en considération des interactions entre observateur et observé, l'observabilité partielle, la détection des comportements trompeurs, l'ajout de contraintes de temps réel, la comparaison de notre algorithme à d'autres approches de reconnaissance de plan, une généralisation du concept de but au sein de notre algorithme ainsi que la conception d'un ensemble de données

étiquetées pour les problèmes de reconnaissance de plan des adversaires.

Grâce à ces améliorations, notre algorithme pourrait être utilisé dans un jeu de stratégie en temps réel pour bien informer les modules de prise de décision quant aux intentions de l'adversaire, afin de prédire ses actions futures et d'exploiter ses faiblesses. L'algorithme pourrait aussi être utilisé dans le cadre de l'assistance aux commandants d'opérations navales pour la tâche d'évaluation de la menace posée par les entités observées dans l'environnement. Certains des concepts que nous avons abordés, en particulier la provocation de l'agent observé, vont au-delà des environnements avec des adversaires et pourraient être utiles par exemple dans les habitats intelligents pour l'assistance à domicile, où les intentions d'un patient doivent être reconnues afin de pouvoir lui venir en aide : la provocation (plutôt l'élicitation d'une réaction) permettrait de demander indirectement aux patients d'expliciter leurs intentions de façon à pouvoir les aider à accomplir leurs tâches.

Annexe A

Plan Recognition Scenarios in RTS Games

In this appendix, we present several scenarios in the RTS domain that illustrate different types of problems that a plan recognition algorithm for that domain would need to solve in order to effectively support a player against his opponents. Players and opponents could either be humans or computer agents. These scenarios in fact convey requirements that an RTS plan recognition algorithm will have to satisfy. The scenarios are general and convey key problems common to most RTS games, such as *StarCraft*, *Age of Empires* and *Command & Conquer*.

A.1 Symbology Conventions

All scenarios presented in this report will use the symbology described below. The observer's entities will use the colour **blue** in figures, and entities that belong to the opponent will use the colour **red**. Moreover, entities that belong to the observer will be subscripted by an $\mathbf{A}$, and those that belong to the opponent will be subscripted by an $\mathbf{E}$. Units will be prefixed by a **U** and structures by an **S**.

While we are mainly interested in two-player games, some scenarios involving $n > 2$ players exhibit problems worthy of note. In these scenarios, the subscripts will be numbered, with $\mathbf{A}_1$ always reserved for the observer, and $\mathbf{A}$ for subsequent allies (with $1 < i \leq n$).

Entities have limited visibility ranges specific to the type of unit or structure, delimited by a dashed line: invisible regions are dimmed. In scenarios where there is no such region, it is understood that the entirety of the map displayed in the scenario figures is visible to the observer. Note, however, that the scenario figures are not necessarily up to scale so the visibility radiuses may vary from one figure to another for a same unit.

While the entities presented below may appear familiar to RTS players, they are not specific to any one game: they may apply to other RTS games and even other similar adversarial domains. The same thing applies to the scenarios that will later be presented.

A.1.1 Units

The units that will be used in the scenarios are presented below. The colour **black** is used to indicate the neutrality of the units' alignments; that is, both parties (allies and opponents) have the ability to recruit these units to constitute their army. All unit symbols sport a short line indicating their current direction.

Infantry Unit

The infantry unit (Figure A.1) is the basic attack unit in a player's army. This unit has mediocre firepower (it is slightly more effective when it is part of a group), it does not possess great defensive capabilities and is easily defeated. Moreover, it is only able to attack other ground units but is vulnerable to ground, air and subterranean units. However, the infantry unit's inexpensive production cost and high mobility makes it ideal as a cheap attack unit, for early game scouting purposes, and as cannon fodder.

Figure A.1: Infantry unit

Subterranean Unit

The subterranean unit (Figure A.2) is an advanced, powerful melee attack unit with the ability to burrow and hide into the ground. When it is burrowed, the subterranean unit cannot be detected nor attacked, except when the opposing force has researched the technology to do so[1]. Moreover, it is only able to attack when it is hidden in the ground: it attacks its enemies (ground units only) by surprise from underneath. After attacking, it must wait a certain amount of time (i.e., a *cooldown* period) before being able to burrow once again. It moves rather slowly but its attack is tremendously powerful and can easily defeat even the most heavily armoured opponents. The subterranean unit is a formidable asset to a player's army.

Figure A.2: Subterranean unit

Flying Unit

The flying unit (Figure A.3) is an advanced airborne attack unit with the ability to attack ground, air and (unburrowed) subterranean units. It is very expensive to recruit but very powerful. It can rather easily be defeated by anti-air units and defensive structures, but its ability to move freely and rapidly as well as its great firepower make it a fearsome unit in a player's army.

Figure A.3: Flying unit

1. This technology requires several advanced structures and is thus generally not researched until rather late into the game.

Heavy Infantry Unit

The heavy infantry unit (Figure A.4) is a lightly-armoured but very powerful ground unit. It is also rather costly to recruit and moves somewhat slowly. It is similar in strength to the subterranean unit but does not need to stop (nor to hide into the ground) in order to attack. Despite its relatively slow speed (when compared to faster units such as the infantry unit and the flying unit), its high recruitment cost and its low defensive capabilities, it is still sufficiently mobile and its tremendous firepower more than makes up for the other disadvantages.

Figure A.4: Heavy infantry unit

Dropship

The dropship (Figure A.5) is an airborne utility unit that has the ability to transport up to eight (8) ground and subterranean units. It allows the fast deployment of slow but powerful units anywhere on the map. It is very heavily armoured and thus very difficult to defeat, but it lacks the ability to attack. Dropships are often used in groups so as to massively deploy ground units. A player observing enemy dropships coming its way should be more concerned with preparing a defence than with assembling anti-air units to defeat it mid-air.

Figure A.5: Dropship

A.1.2 Structures

The structures that will be used in the scenarios are presented below. The colour **black** is used to indicate the neutrality of the structures' alignments: that is, both parties (allies or opponents) have the ability to build these structures.

Command Centre

The command centre (Figure A.6) is the main structure of a player. It provides some basic abilities (infantry units recruiting, resources acquiring, and maximum population limit increasing) upon which rely other, more advanced, abilities in the game. As such, the command centre is a very valuable structure for a player to defend and, conversely, the opponent's command centres are key structures to destroy.

Figure A.6: Command centre

Bunker

The bunker (Figure A.7) is a defensive structure that can hold up to four (4) units. While inside the bunker, these units will be protected from enemy fire and can use their basic attacks (i.e., special attacks cannot be used) against opponents.

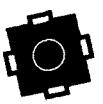


Figure A.7: Bunker

Cavern

The cavern (Figure A.8) is an advanced structure that allows a player to recruit subterranean units. Since it is the sole purpose of this structure, a player will generally not build it unless it intends to recruit this type of unit; it is thus a clear indicator of a player's intentions.

Spire

The spire (Figure A.9) is an advanced structure that allows a player to recruit flying units. Since it is the sole purpose of this structure, a player will generally not

Figure A.8: Cavern

build it unless it intends to recruit this type of unit: it is thus a clear indicator of a player's intentions.

Figure A.9: Spire

A.2 Deception

The objective of this scenario is to demonstrate the problems encountered by an observer confronted with an opponent using deception. That is, the opponent is deliberately executing actions to mislead the observer into thinking he is trying to achieve some other objective. Such behaviours are generally hard to recognize properly by a human, let alone a computer AI. Yet, being able to recognize them provides an advantage over an opponent unable to do it as well, recognizing these plans.

Figure A.10 shows the opponent **E** deceiving the observer **A** into thinking that he is following a strategy oriented on recruiting subterranean units when he is actually concentrating on recruiting flying units (remember that these strategies are usually mutually exclusive). **E** thus expects **A** to prepare defences against the subterranean units, neglecting to prepare an anti-air defence.

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

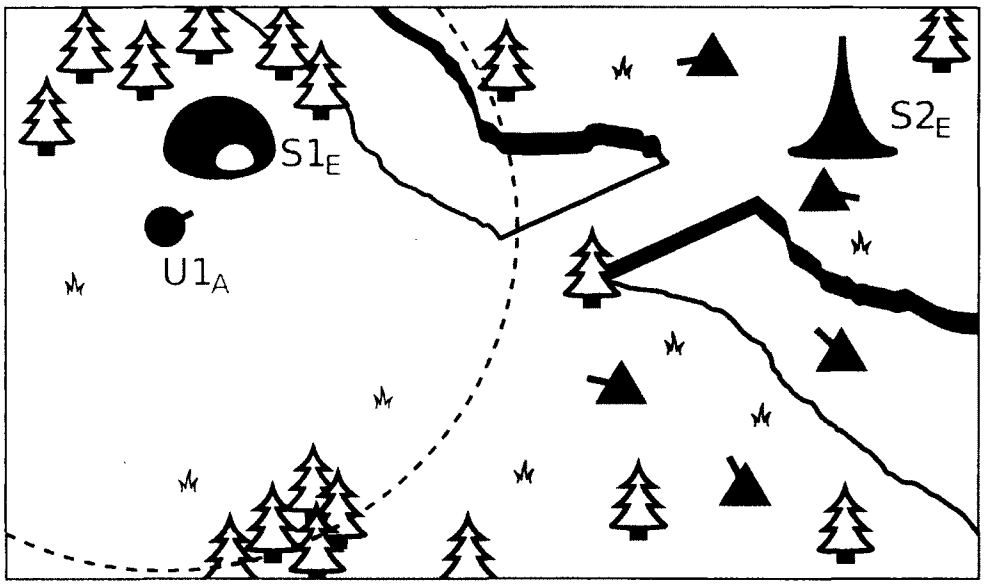


Figure A.10: Deception scenario

- ***E** builds a Spire (**S2**$_E$) and starts recruiting flying units;*
- ***E** builds a Cavern (**S1**$_E$) outside its base;*
- **A** sends a scout (**U1**$_A$) to explore the map;
- **U1**$_A$ notices a Cavern and concludes that **E** is planning to recruit subterranean units; and
- **A** starts preparing an anti-subterranean unit defence.

Opponent's Rationale

E is trying to get the edge over **A** by leading him to believe he will recruit (or has already started recruiting) subterranean units, when he actually has already recruited flying units. **E** expects **A** to recruit anti-subterranean units and thus neglect an anti-air defence, which would allow **E** to easily defeat **A**. In order to carry out this tactic, **E** decides to build a Spire in a remote, heavily defended area so that it will not easily be discovered by **A**. On the other hand, **E** purposely builds a Cavern in a location that **A** is likely to explore. By the time **A** would discover the Cavern, **E** would already have an army of flying units ready to attack and defeat **A**, defenceless against airborne

units.

Observer's Rationale

A sends a scout to explore the map in order to maintain an up-to-date awareness of the situation. Upon discovering the enemy Cavern, **A** concludes that **E** is going to start (or has already started) recruiting subterranean units, since observing a Cavern usually is a clear indicator that the opponent is committed to a subterranean unit-recruiting strategy. Knowing this, **A** decides to start mounting an anti-subterranean defence instead of wasting resources on an anti-air defence.

Algorithmic Problem

The observer should have been suspicious of the enemy Cavern's location and its lack (or low number) of defensive structures and units. Moreover, the fact that **A** has still not been attacked by subterranean units so far in the game might also be an indicator that the opponent's strategy is not oriented on recruiting subterranean units after all: something fishy might be going on. Yet, failing to recognize the opponent's true intent of flying units is fatal for the observer.

The problem here is to accurately identify the opponent's intention given the available information, that is, the presence of an undefended (or lightly defended) enemy Cavern. As pointed out by Elsaesser et Stech [29], deception works because it exploits reasoning errors, cognitive limitations and concomitant biases. In this scenario, **E** managed to successfully deceive **A** for two reasons: (1) **A** only had access to a limited amount of information (i.e., he did not know that **E** had also built a Spire); and (2) **A** jumped to conclusions using only this incomplete information. Indeed, even if a player normally does not have any incentive to build a Spire if he does not plan to recruit subterranean units, other possibilities (such as a deception tactic or a simple mistake) are not invalidated and should not be discarded preemptively nor ignored entirely.

A.3 Diversion

The objective of this scenario is to underline the need for a system capable of recognizing diversions, i.e., tactics used to draw the enemy away from the primary target.

Figure A.11 shows the opponent **E** deploying empty dropships towards the primary base ($\mathbf{S1_A}$) of the observer **A**. However, **A** has no way of knowing whether the dropships are currently transporting powerful units or not; he can only assess the threat they would pose if the dropships were not empty. Therefore, **A** decides to move his troops towards $\mathbf{S1_A}$ to defend it from possibly incoming enemy units, leaving $\mathbf{S2_A}$ undefended and vulnerable. Once the path is clear, $\mathbf{U2_E}$ moves out to attack $\mathbf{S2_A}$.

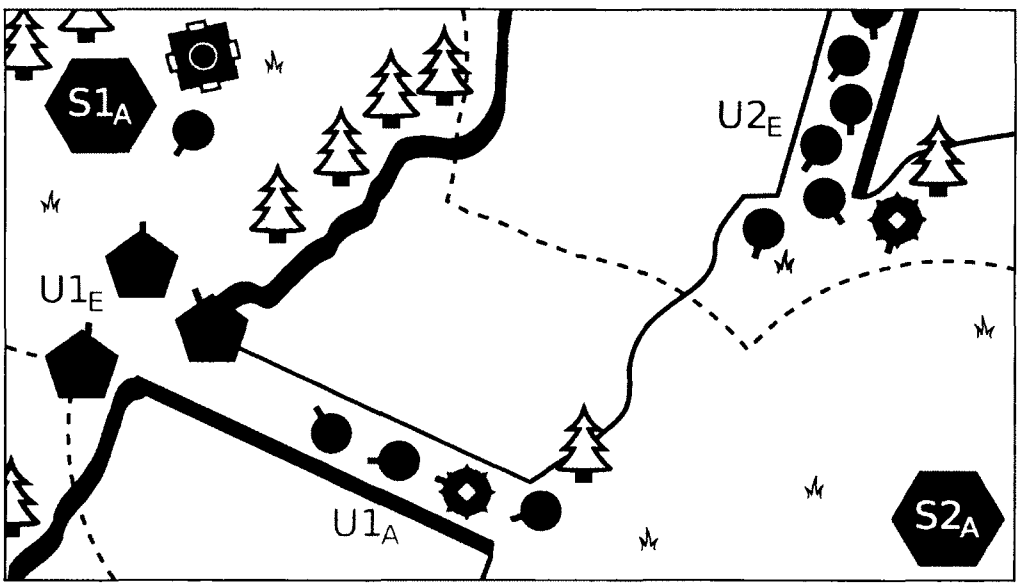


Figure A.11: Diversion scenario

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

- ***E** sends infantry units ($U2_E$) towards the observer's secondary base ($S2_A$);*
- ***E** sends empty dropships ($U1_E$) towards the observer's main base ($S1_A$);*

- *The infantry units* $\mathbf{U2}_E$ *stop at the south end of the northeastern bridge*:
- **A** detects **E**'s dropships $\mathbf{U1}_E$ approaching its main base $\mathbf{S1}_A$:
- **A** moves his units ($\mathbf{U1}_A$) to defend $\mathbf{S1}_A$ from the incoming enemies expected to be deployed from the dropships:
- The dropships $\mathbf{U1}_E$ keep hovering over $\mathbf{S1}_A$: and
- The infantry units $\mathbf{U2}_E$ attack **A**'s undefended secondary base $\mathbf{S2}_A$.

Opponent's Rationale

The opponent would like to gain an advantage over **A** by destroying his secondary base $\mathbf{S2}_A$, but lacks the opportunity to do so since it is defended, and attacking it would result in massive casualties for **E**. Therefore, **E** devises a plan that will make **A** move out his units guarding $\mathbf{S2}_A$. To do so, **E** sends empty dropships towards $\mathbf{S1}_A$ to make **A** believe that a large amount of powerful units will be deployed there. As a result, **E** expects $\mathbf{U1}_A$ to be driven away from $\mathbf{S2}_A$ to help defend $\mathbf{S1}_A$, thus clearing the way and enabling **E** to attack $\mathbf{S2}_A$. For **E**, the cost of employing empty dropships (even if they had to be recruited first) to bait **A** is lower than the cost of the casualties that would surely result from trying to attack a guarded base.

Observer's Rationale

A player's main base is often more important to defend throughout the game than a secondary (and tertiary, and so on) base. The reason for this is that the first base is often the chosen location for structures used to recruit advanced units, research technological upgrades, etc. While good players might be able to recover from the loss of one's main base, this situation is still highly undesirable.

For this reason, the observer decides to send his units to defend the main base $\mathbf{S1}_A$ rather than the secondary base $\mathbf{S2}_A$. As a result, $\mathbf{S2}_A$ is left undefended and vulnerable, but this is still somewhat less undesirable than risking to lose the main base where several enemy units are expected to be deployed from the dropships. Had **A** been able to detect the mass of units ($\mathbf{U2}_E$) waiting to attack $\mathbf{S2}_A$, he would have been able to conclude that either the dropships are but a diversion, or that the opponent is planning to attack both bases at the same time (an estimate of the

opponent's assets and resources could be computed to assess the plausibility of this statement).

Algorithmic Problem

This scenario highlights the requirement for active observation on the observer's part, since the information obtained from the opponent's actions alone is not always sufficient to infer his intentions. The indicators available to the observer concerning the opponent's behaviour are clearly not enough to conclude that the dropships are empty, nor that an attack on $\mathbf{S2}_{\mathbf{A}}$ is being prepared. It is nonetheless a possibility that should not be discarded by the plan recognition algorithm. Its low likelihood, coupled with the highly undesirable outcome, could perhaps let the observer know that the opponent has something fishy in mind.

A.4 Baiting

The objective of this scenario is to show a situation where the opponent is baiting the observer to make him move away from a certain region. It is actually a variation of the Diversion scenario presented in Section A.3.

Figure A.12 depicts the opponent **E** waiting for an opportunity to attack the main base ($\mathbf{S1}_{\mathbf{A}}$) of the observer **A**. The base is surrounded by a forest with two openings. To bait **A**, **E** plans to send one unit through the eastern opening, wait for it to be attacked by **A** and then move the remaining units through the western opening to attack the now undefended base $\mathbf{S1}_{\mathbf{A}}$.

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

- ***E** sends his units ($\mathbf{U1}_{E}$) towards **A**'s base ($\mathbf{S1}_{A}$);*
- *The group of units $\mathbf{U1}_{E}$ arrives to the forest surrounding $\mathbf{S1}_{A}$;*
- **E** moves one unit inside the region controlled by **A** through the eastern opening of the forest;

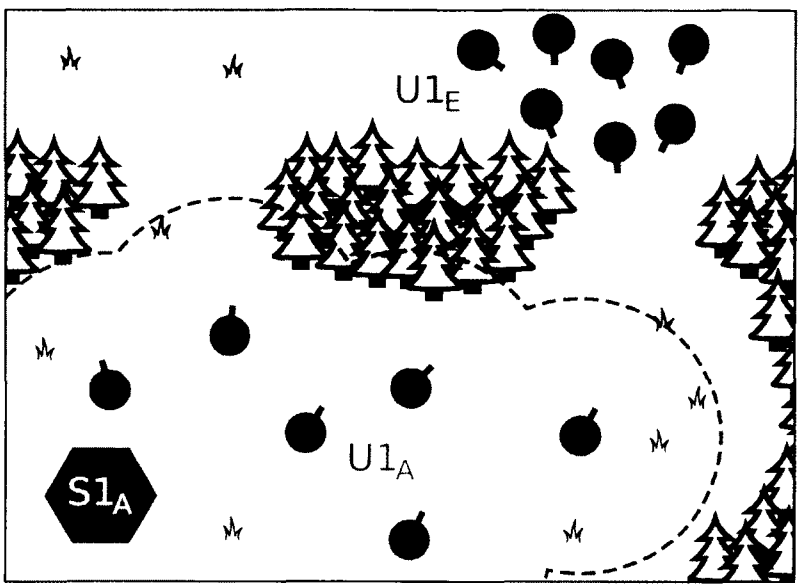


Figure A.12: Baiting scenario

- **A** moves his units ($\mathbf{U1_A}$) to attack the invading enemy;
- **E** moves his remaining units $\mathbf{U1_E}$ through the western opening; and
- The group of units $\mathbf{U1_E}$ attacks the undefended base $\mathbf{S1_A}$.

Opponent's Rationale

This scenario is quite similar to the one in Section A.3, so the opponent's rationale is also very similar. The opponent's objective in this scenario is to attack the observer's main base ($\mathbf{S1_A}$), which is expected to be guarded. **E** sends a unit for two reasons: (1) to scout the region and learn about **A**'s whereabouts and defensive capabilities; and (2) to clear the path to attack the base. While the opponent does not know what to expect on the other side of the forest any more than the observer does, he is willing to sacrifice a unit. Indeed, losing only one unit is a much better outcome than blindly moving a group of units into enemy territory, where defensive structures and units are likely awaiting. Of course there is always the risk that **A** is not going to fall for it and only move one unit to kill the intrusive scout, if at all, but that is a risk **E** is willing to take, given the expected favourable outcome of a

successful baiting tactic.

Observer's Rationale

From the observations available to **A**, it is impossible to predict that **E** is up to something more sinister than simply scouting the region. **A** thus decides to eliminate the invading scout as soon as possible with all units from the $\mathbf{U1_A}$ group, to make sure the scout does not have the time to gather any intelligence on the structures and technology possessed by **A**. This results in clearing the western opening in the surrounding forest to the following attack by **E**, which **A** has no way of predicting from the available data.

Given **A**'s entities as presented in the scenario, there is not much the observer could have done differently to avoid exposing itself to an attack, except for only commanding one or two units to attack the enemy scout. Observing this, however, the opponent could very well have retorted by moving all his remaining units at once through the eastern opening, taking **A**'s attacking units off-guard and killing them, making **E**'s group of units more populous than **A**'s remaining units. **E** would then probably manage to eliminate the rest of $\mathbf{U1_A}$ and destroy $\mathbf{S1_A}$, perhaps with some casualties.

Algorithmic Problem

The problem highlighted by this scenario is the same as in the previous scenario, that is, the observer has no way to predict the opponent's following course of action given the observations obtained so far. The highly undesirable outcome of the opponent's successful use of a baiting tactic, however, could perhaps be used as a cue to predict this behaviour. However, the need for active observation is clear and evident: the observer must regularly scout the map (among other things) in order to maintain a sufficient level of situation awareness and to avoid being caught off-guard by the opponent.

A.5 Provocation – Ambush

The objective of this scenario is to present a situation where the observer provokes the opponent, expecting him to react in a certain way that will help disambiguate his true intentions. Provoking an opponent and observing his reaction can be seen as a method an agent can use to help solve the problem of plan recognition in an adversarial setting, rather than a plan recognition challenge in itself.

Figure A.13 shows a certain number of enemy units attacking the observer's secondary base ($\mathbf{S2}_\mathbf{A}$). After realizing that he is under attack, **A** moves some of his units ($\mathbf{U1}_\mathbf{A}$) to defend $\mathbf{S2}_\mathbf{A}$. **E** then retreats his units ($\mathbf{U1}_\mathbf{E}$) back through the eastern bridge, hoping that **A** will follow and walk right into a trap set earlier on the other side of the bridge. One way for **A** to find out whether **E** is retreating in order to lure him into a trap or simply to save itself is to have only one unit follow $\mathbf{U1}_\mathbf{E}$; if the unit is attacked, this confirms that **E** planned to lure **A** into a trap, otherwise $\mathbf{U1}_\mathbf{E}$ will keep trying to evade the unit.

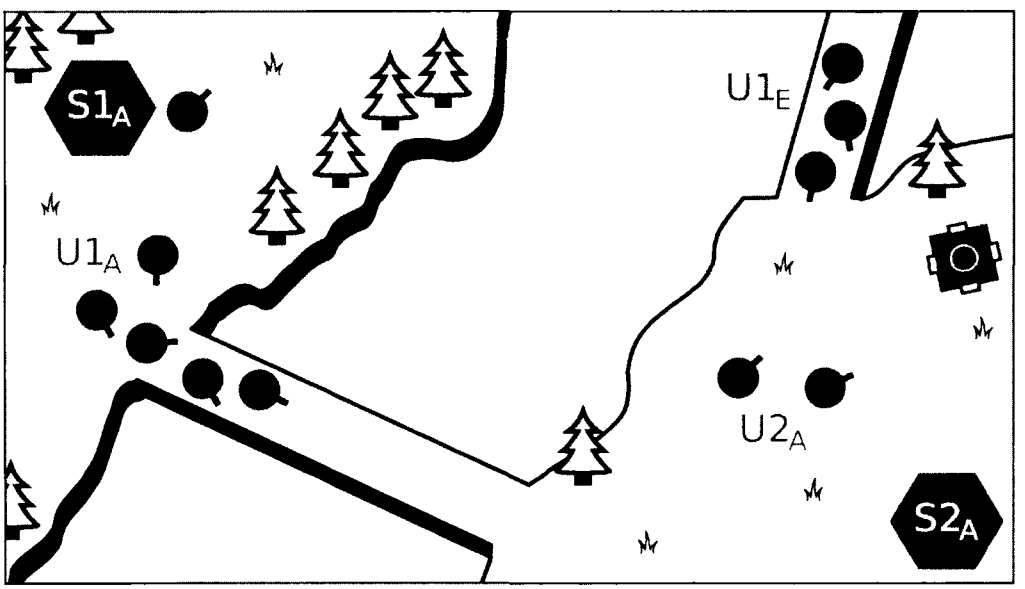


Figure A.13: Provocation scenario

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

- ***E** moves several subterranean and infantry units (**U1**$_E$) towards the eastern bridge*;
- ***E** makes the subterranean units burrow into the ground at the northern end of the bridge*;
- The infantry units (**U1**$_E$) move towards the observer's secondary base (**S2**$_A$);
- **U1**$_E$ starts attacking **U2**$_A$;
- **A** moves its units **U1**$_A$ towards **S2**$_A$ to help defend it;
- **U1**$_E$ retreats through the eastern bridge;
- **U1**$_E$ stops *once it joined with the burrowed subterranean units*;
- **A** moves one unit to chase after **U1**$_E$; and
- **E** attacks the unit instead of keeping escaping, [involuntarily] revealing its intentions.

Opponent's Rationale

Similarly to the two previous scenarios, the opponent is trying to lure **A** somewhere else. This time, however, **E** is simply hoping that **A** will move all its units at once so that they can all be killed from underneath by the burrowed subterranean units. To achieve this stratagem, **E** moves some of his units to attack **A**'s secondary base (**S2**$_A$), expecting that several units will come to the rescue. Two things, both desirable for **E**, may happen here: (1) **A** does not send any units to defend **S2**$_A$ (either because he is not able to do so or because he does not want to) and **E** destroys the structure; or (2) **A** moves a certain number of units to help defend **S2**$_A$. If the second option were to occur, **E** would retract through the eastern bridge hoping to lure **A** right into the awaiting burrowed subterranean units.

Observer's Rationale

The observer does not possess the technology required to observe and engage burrowed subterranean units. Moreover, he did not gather any indicators that could lead him to believe **E** has such units composing his army. When **A** observes enemy units attacking his secondary base ($\mathbf{S2_A}$), he decides to move units there in order to help defend it, without really giving any further thoughts about it. A player's command centre is a very valuable structure to defend, since it allows its owner to increase resource gathering and unit production, and also provides a new location from where to deploy troops to attack the opponent.

When the invading units ($\mathbf{U1_E}$) retreat even before they are attacked, **A** wonders what could be the reason for **E**'s behaviour. One possibility is that he feels outnumbered and prefers to save his units rather than losing them without wreaking much havoc. Another possibility is that **A** is being lured into a trap. To find out **E**'s true intentions, **A** can provoke his opponent, for example by having only one or two units chase after the escaping $\mathbf{U1_E}$. This would force **E** to react to **A**'s actions, revealing at the same time his intentions.

Algorithmic Problem

Since the observer does not know the opponent's true intentions, he needs to somehow obtain the missing information that would allow him to better understand the opponent's behaviour. Otherwise, the observer will be confined to wait for the opponent's next action, which could potentially be disastrous. Provoking incites the opponent to act in reaction to the observer's own actions. The problem here is to correctly decide *when*, *if* and *how* to provoke the opponent. The algorithm needs to decide when a provocation would be beneficial for the plan recognition inference process, if it is actually worth doing, and how should it be carried out.

Note that the current scenario does not account for the fact that the opponent might be aware of the observer's train of thought, that is, he knows that he is only moving some units to assess the presence of the trap. Indeed, humans and game-theoretic agents recursively model their opponent's mind and vice versa, that is, "I think that the opponent thinks that I think that he thinks that ...". The problem

of selecting a provocation and interpreting the reaction (if any) should thus take into consideration the fact that the opponent might adapt his behaviour if he suspects that he is being set up by the observer.

A.6 Provocation – Containment

The objective of this scenario is also to present a situation where the observer watches for the opponent's reaction to a provocation, in order to help disambiguate his intentions. Without provoking the opponent, the observer would be left in the dark until it is too late to respond to the opponent's attack.

Figure A.14 shows the opponent waiting outside the observer's base, blocking the only entry point (the *chokepoint*). The dimmed red region in the figure is invisible to the opponent; the observer can see everything visible on the figure. From the observer's perspective, it is unclear why the opponent is just sitting there: is he waiting for more units to join and launch an attack? or is he simply blocking the agent from exiting his own base[2]? A plan recognition algorithm passively waiting to observe the opponent's actions would be unable to understand and explain the opponent's behaviour until he eventually launches an attack.

Scenario Events

In this scenario, the following events occur:

- **E** moves some units ($\mathbf{U1}_E$) towards **A**'s base;
- The group of units $\mathbf{U1}_E$ stops at the eastern end of the bridge;
- **A** moves some units ($\mathbf{U1}_A$) towards $\mathbf{U1}_E$;
- $\mathbf{U1}_A$ and $\mathbf{U1}_E$ start attacking one another; and
- **A** provokes **E** by having $\mathbf{U1}_A$ retreat inside the base.

From this point on, there are two possible reactions, depending on the opponent's true intentions. Both outcomes will be discussed below.

- **E** follows $\mathbf{U1}_A$, confirming the attack hypothesis: **OR**

2. Containing the enemy within his own base allows an agent to assert map control while the enemy is denied the capability to move out any units to scout the map, build an expansion, etc.

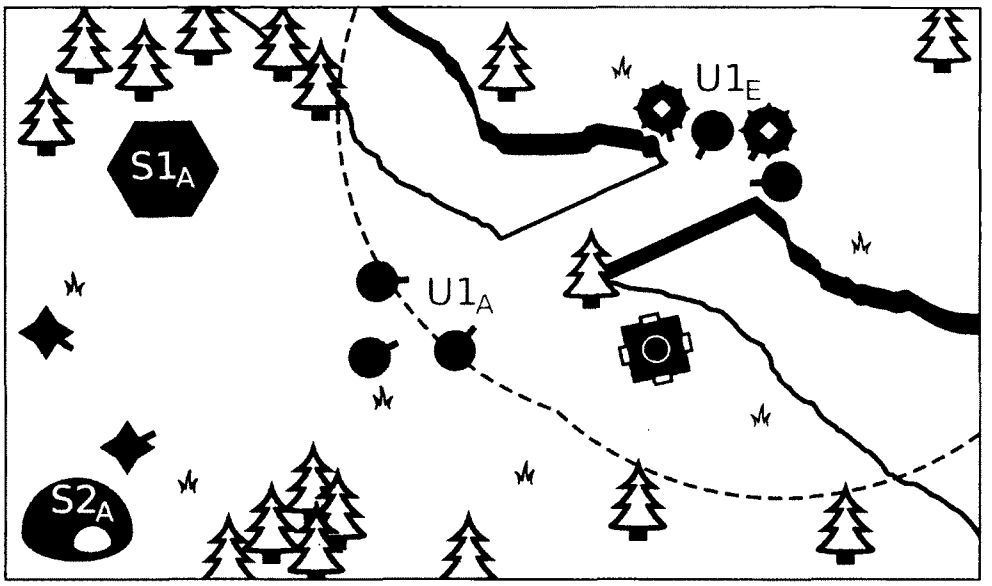


Figure A.14: Provocation scenario

– **E** does not follow $\mathbf{U1_A}$, confirming the containment hypothesis.

Opponent's Rationale

The next two subsections describe the opponent's rationale if either hypothesis were the true one.

Attack Hypothesis

The opponent's objective is to launch an all-out attack into **A**'s base. However, **E** is waiting for the right opportunity to attack, as well as a sufficient capability to do so, which includes having a superior force. For this reason, the opponent is simply waiting for more attack units to join in order to start the attack. The units currently stationed at the chokepoint also serve as reconnaissance units, providing the opponent with an up-to-date assessment of the local tactical situation.

When the observer decides to retreat after attacking the stationed units, **E** believes that **A** had overestimated his own force prior to the attack and thus retreated inside the base upon realizing his mistake. Then, **E** interprets this as a cue that his stationed

units constitute a sufficient force to invade the base, and hence sends his units to chase after the observer's retreating units.

Containment Hypothesis

The opponent's objective is to deny to the observer the capability to move out any units to scout the map, build an expansion, launch an attack, etc. At the same time, a containment allows **E** to simultaneously pursue and accomplish these objectives without being threatened by **A**'s presence.

When **A** retreats after attacking the stationed units, **E** does not follow them since this action would break the containment barrier, the primary objective of this strategy. Indeed, chasing after the retreating units would allow **A** to slip past the breached blockade, for example to build defensive towers, or to attack the opponent's units in charge of building new expansions (who are rather poorly defended, since a successful containment strategy voids the need to defend anything, the enemy being held hostage in its own base). This is what **E** is trying to avoid at all costs: hence, the stationed units remain in place to make sure no enemy unit can escape.

Observer's Rationale

The observer observes a number of enemy units sitting just outside his base, but is unable to identify the reason why the opponent is waiting there. **A** can, however, formulate hypotheses that explain **E**'s behaviour. The first hypothesis states that the opponent is simply waiting for reinforcements to join before launching an all-out attack into the base. If this explanation turns out to be the right one, **A** should build defensive structures and mount a defensive force to kill the invaders before they get inside the base. However, preparing such a defence is completely useless if the second hypothesis, which states that the opponent opted for a containment strategy, is the correct one. In this situation, **A** should invest all efforts in breaking the blockade as fast as possible to make sure **E** does not take the economic lead, which is the long-term effect sought by a containment strategy.

For these reasons, the observer needs to disambiguate the opponent's true intentions to decide on the correct course of action. To do so, **A** moves some units to attack

the opponent's and then has them retreat before they get killed. As mentioned earlier, on opponent executing a containment will do anything to avoid a breach in the blockade, whereas an opponent simply waiting to have the superior force to attack might interpret the agent's decision to retreat as an indication that he feels overwhelmed by the opponent.

Algorithmic Problem

The problem in this scenario is the same as the one highlighted in the previous scenario, that is, being able to decide when, if, and how to provoke the opponent. The plan recognition algorithm needs to be able to anticipate the opponent's behaviour (i.e., the reaction to the provocation) and trick him into revealing his true intentions in a predictable way (so as not to increase the uncertainty on the opponent's intentions). Again, the algorithm should take into account the fact that the opponent might realize that he is being tricked, and adapt his behaviour accordingly.

A.7 Multiple Goals

The objective of this scenario is to show a situation where the observer is confronted with an opponent having multiple concurrent goals. More specifically, the opponent is executing some actions that serve two purposes, or two plans, simultaneously. This can be difficult for some plan recognition algorithms, yet this is certainly not an unusual situation in the RTS domain.

Figure A.15 shows the observer observing an enemy unit moving towards the bridge, and then disappearing. However, this unit's moving in this direction serves two purposes: (1) scouting the area; and (2) building a bunker ($\mathbf{S1}_{\mathbf{E}}$). Understanding the intentions of the opponent in this situation is difficult, since the actions he executes are not necessarily motivated by only one unique objective. In order to correctly recognize the opponent's plans from the observations, this aspect has to be accounted for by the algorithm.

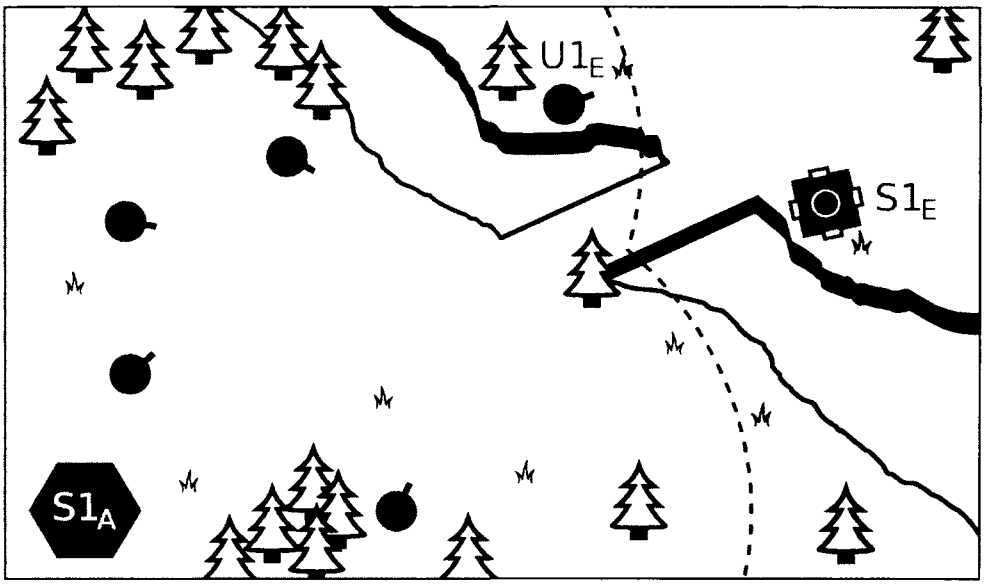


Figure A.15: Multiple goals scenario

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

- **E** sends a unit (**U1**$_E$) towards **A**'s base to scout the area;
- **A** observes **U1**$_E$ walking around;
- **U1**$_E$ keeps moving towards the bridge and then disappears from **A**'s visibility region;
- ***U1***$_E$ *builds a bunker (****S1***$_E$*) next to the bridge*; and
- ***U1***$_E$ *gets inside the bunker to attack the units who would cross the bridge.*

Algorithmic Problem

The opponent's rationale is rather simple to understand for a human: he simply wants to scout a region and, at the same time, build a defensive structure to attack enemy units that may pass near it. This is more problematic for a plan recognition algorithm not designed to handle multiple concurrent goals. Indeed, even if such an algorithm could generate several "one goal-only" hypotheses to explain the observa-

tions, the hypothesis stating that the opponent is simply scouting would most likely have a better rank (e.g., a larger conditional probability in a probabilistic system) than the hypothesis stating that the opponent is going to build a bunker, since the actual action of building the structure has not been observed yet, while the action of moving around has. Either way, the algorithm would still fail to correctly identify the opponent's true intentions, that is, pursuing two objectives simultaneously.

A.8 Coordination

The objective of this scenario[3] is to present a situation where enemy units coordinate themselves and cooperate in order to achieve broader objectives than those individually pursued by each unit. Humans are experts at generalizing facts and performing abstract reasoning, for example considering groups of units as abstract entities instead of looking at each unit individually. However, such an ability to formulate abstractions is more difficult for AI systems. Note that this scenario could also involve groups of units that belong to two or more different opponent agents.

Figure A.16 shows two groups of enemy units ($\mathbf{U1}_E$ and $\mathbf{U2}_E$) who coordinate each other in order to perform a pincer attack on the observer. Each group of units uses a bridge in order to surround the observer's units ($\mathbf{U1}_A$) on the other side of the river.

Scenario Events

In this scenario, the following events occur:

- The opponent moves a group of units ($\mathbf{U1}_E$) through the southern bridge towards the observer's units;
- *[Simultaneously]* The opponent moves a group of units ($\mathbf{U2}_E$) through the northern bridge towards the observer's units;
- Both groups reach the other side of the river and surround $\mathbf{U1}_A$; and
- $\mathbf{U1}_E$ and $\mathbf{U2}_E$ defeat $\mathbf{U1}_A$ using a pincer attack.

3. This scenario was originally authored by Julien Filion.

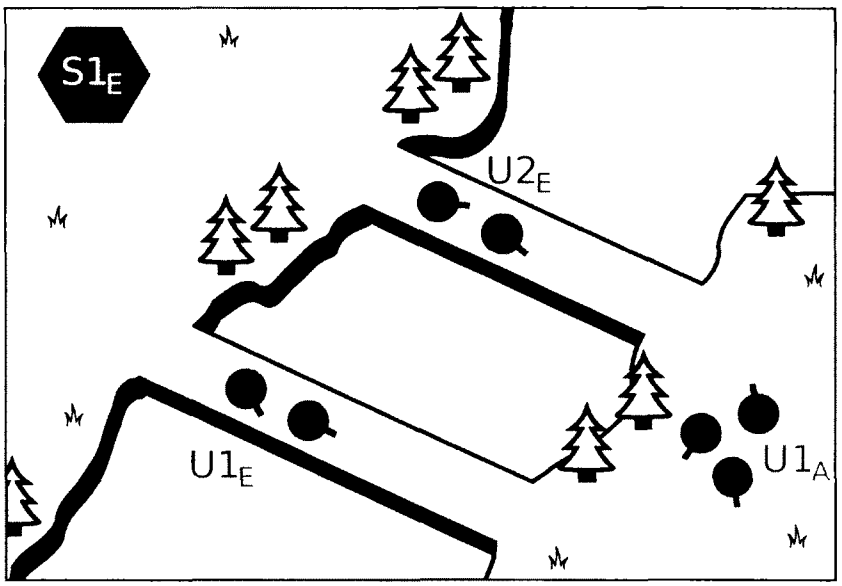


Figure A.16: Coordination scenario

Algorithmic Problem

A plan recognition algorithm that is not specifically designed to recognize coordinated group behaviours will probably make the assumption that actions observed from different units are independent from one another. This assumption would not be problematic until the opponent started to engage in coordinated behaviours, that is, until the opponent had each unit achieve some small objective that, when all combined, accomplish something greater.

In this scenario, the behaviour of each unit, when considered independently, could be interpreted as inoffensive scouting before the actual attack, and thus deemed not threatening by the plan recognizer. If the observer were able to figure out that a pincer attack was being prepared by the opponent, however, it could start by eliminating the units at the chokepoint of one of the bridges and then finishing off the remaining units, having gained numerical superiority.

A.9 Cyclic Behaviour

The objective of this scenario is to present another situation where a group of units coordinate their actions and behaviours, this time in a cyclic manner. The plan recognition algorithm should be able to understand the cyclic nature of the behaviour, and thus to predict that the same (of very similar) actions will eventually be repeated.

Figure A.17 shows the opponent using a fast dropship ($\mathbf{U2_E}$) to rapidly drop slower heavy infantry units ($\mathbf{U1_E}$) and pick them back up right after they have attacked. This allows the opponent to deploy a slow but heavy attack force and keep it safe from attackers using a fast airborne transportation utility unit.

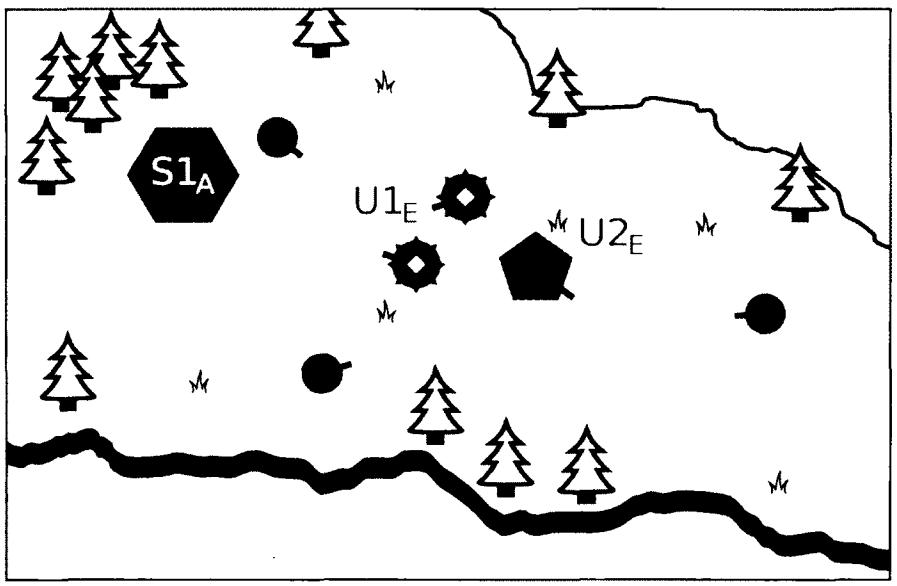


Figure A.17: Cyclic behaviour scenario

Scenario Events

In this scenario, the following events occur. Events undetected by the observer are italicized.

- ***E** embarks two heavy infantry units ($\mathbf{U1_E}$) abord a dropship ($\mathbf{U2_E}$);*
- **E** sends $\mathbf{U2_E}$ towards **A**'s base;
- $\mathbf{U2_E}$ unloads $\mathbf{U1_E}$ into **A**'s base and circles around the area;

- The heavy infantry units $\mathbf{U1_E}$ attack $\mathbf{A}$'s base ($\mathbf{S1_A}$):
- Before $\mathbf{A}$'s units can start attacking the invaders, $\mathbf{U2_E}$ picks up $\mathbf{U1_E}$:
- $\mathbf{U2_E}$ moves away from $\mathbf{A}$'s base, hoping to escape $\mathbf{A}$'s units; and
- $\mathbf{U2_E}$ comes back to $\mathbf{A}$'s base, unloads $\mathbf{U1_E}$ and so on.

Algorithmic Problem

While it is not difficult to understand that the opponent's goal is to harass and attack the observer, recognizing *how* it intends to do so (and how the observer can circumvent the attack) is less obvious for a computer AI. Indeed, an effective plan recognition algorithm should be able to infer that it is more likely that the opponent is going to repeat the same attack pattern indefinitely, rather than try any another tactic. This is the key to properly understand and counter this tactic, and observing any change in the cyclic behavioural pattern should be interpreted as an important hint that something is off.

A.10 Collaboration – Inference Conflicts

The objective of this scenario[4] is to highlight a problem that arises in collaborative plan recognition, that is, several players who cooperate to solve the plan recognition problem.

Figure A.18 shows two players ($\mathbf{A_1}$ and) who collaborate to observe and understand the actions of the opponent $\mathbf{E}$. The dimmed green area is invisible to and the dimmed blue area is invisible to $\mathbf{A_1}$, as before. Thus, enemy unit $\mathbf{U1_E}$ is visible to both $\mathbf{A_1}$ and , but $\mathbf{U2_E}$ is only visible to .

Algorithmic Problem

The main challenge that arises from this scenario is the problem of conflictual inferences the observer and his ally may make. Indeed, since $\mathbf{A_1}$ only sees $\mathbf{U1_E}$, he could conclude that the unit's objective is to scout the area and perhaps build a

4. This scenario was originally authored by Gabriel Beaulieu.

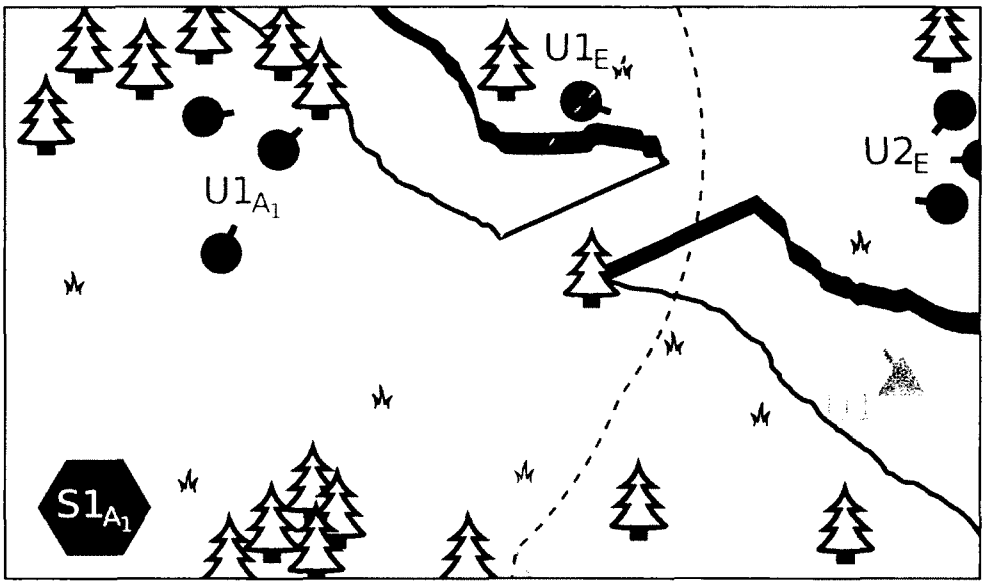


Figure A.18: Collaboration scenario

defensive structure at the end of the bridge, much like the Multiple Goals scenario in Section A.7. , who has more available information, could conclude that the opponent intends to attack the observer's base ($S1_{A_1}$) with both groups of units. Thus, even if both players use exactly the same plan recognition algorithm, they may still explain the opponent's action differently if they don't share any information.

One information sharing scheme might have each observer run his own inference process, and then share the outcome with his allies. Since conflicts are possible, agents would then have to debate and argue with one another to resolve any conflicts, either in a centralized or decentralized fashion.

A possibly more robust information sharing scheme might have agents share observations with one another (or with a centralized "server") as soon as they are obtained. However, one could imagine several problems with this approach, such as the obviously large amounts of data that would need to be shared among agents. This problem could somewhat be mitigated by avoiding to share observations that were already acquired by other agents. For example in this scenario. would not need to tell A_1 about $U1_E$. Determining if the information was already obtained is a problem in itself, and it could exacerbate another problem, that is, how to know if an action

was executed twice (or more often) or if it is a duplicate that can safely be ignored.

A.11 Collaboration

The objective of this scenario[5] is to motivate collaborative plan recognition by showing how it can allow players to obtain more information about the opponent's whereabouts and actions than they could if they did not have allies.

Again, Figure A.19 shows players $\mathbf{A_1}$ and collaborating to observe and understand the actions of the opponent **E**.

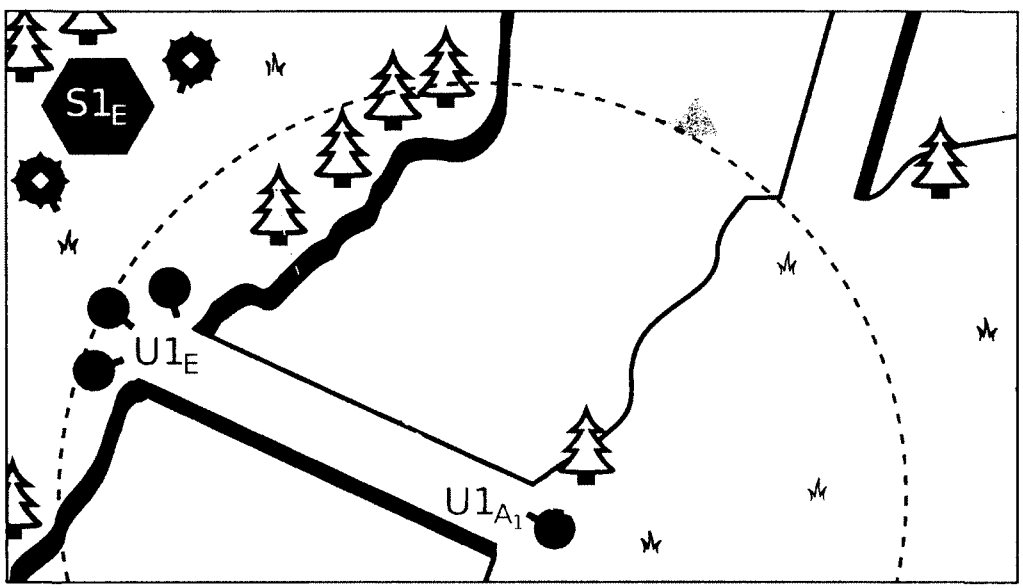


Figure A.19: Collaboration scenario

Scenario Events

In this scenario, the following events occur. Events unobserved by the observer $\mathbf{A_1}$ and his ally are italicized.

- ***E** moves some units (**U1E**) to block the western bridge;*
- $\mathbf{A_1}$ sends a unit ($\mathbf{U1_{A_1}}$) to scout **E**'s base;

5. This scenario was originally authored by Gabriel Beaulieu.

- ***E*** *recruits heavy infantry units in his base, undisturbed by potential invaders thanks to the bridge blockade*;
- $\mathbf{U1_A}$ is blocked by $\mathbf{U1_E}$ guarding the other end of the bridge;
- $\mathbf{A_1}$ asks to help him scout **E**'s base; and
- sends a flying unit ( ) towards **E**'s base to scout it from the air.

Opponent's Rationale

The opponent in this scenario simply wants to refrain his enemies from interrupting the recruitment of heavy infantry units and the overall development of his base, as well as to prevent them from scouting it. Hence, **E** decides to block the bridge with some units ($\mathbf{U1_E}$) since it is the only ground-level entry point into the base.

Observer's Rationale

When the observer's scout ($\mathbf{U1_{A_1}}$) notices the opponent's units blocking the other end of the bridge, he has two options for his following course of action. First, he can bring in more units, attack the opponent's blockade, and then resume scouting the opponent's base, provided that the latter does not send any reinforcements (which is rather unlikely). Alternatively, he can ask his ally, , for help in scouting the opponent's base, and then share the relevant information. This would of course be impossible in a two-player game (unless, obviously, $\mathbf{A_1}$ had an available flying unit which, for the sake of this scenario, we will assume is not the case).

Algorithmic Problem

This scenario does not really highlight any problem from an algorithmic point of view. However, a collaborative plan recognizer should consider the observer's allies as other sources of information (auxiliary "sensors" of sorts), from which it is possible to collect additional data concerning the opponent's actions and plans.

A.12 Typical Game

The objective of this rather lengthy scenario is to present an excerpt from a typical RTS game, with an opponent trying to achieve concurrent objectives while reacting to external events, such as the presence of enemy units.

Figure A.20 shows the opponent **E** trying to achieve two main objectives simultaneously: (1) attack and destroy the observer's nearby base and stationed units ($\mathbf{S1_A}$); and (2) build a new command centre ($\mathbf{S2_E}$) to increase the unit production rate, and then defend the newly-built structure. Each of these tasks is performed by a different group of units (respectively, $\mathbf{U1_E}$ and $\mathbf{U2_E}$), and thus the scenario could be considered a combination of two independent sub-scenarios.

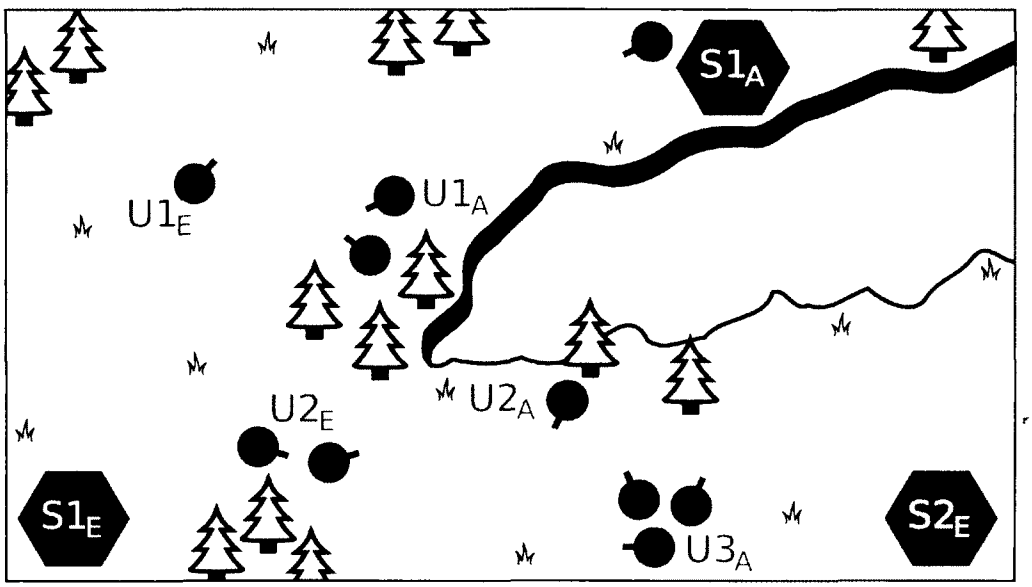


Figure A.20: Typical game scenario

Scenario Events

In this scenario, the following events occur:

- **E** sends a unit ($\mathbf{U1_E}$) towards the observer's base ($\mathbf{S1_A}$);
- **A** observes $\mathbf{U1_E}$ and sends two units ($\mathbf{U1_A}$) to attack it;
- $\mathbf{U1_E}$ evades $\mathbf{U1_A}$ and keeps moving towards $\mathbf{S1_A}$;

- $\mathbf{U1_E}$ reaches $\mathbf{S1_A}$ and starts attacking it and the defending unit;
- $\mathbf{U1_E}$ moves away from $\mathbf{S1_A}$ after a while to avoid being attacked; and
- $\mathbf{U1_E}$ restarts attacking and moving away from $\mathbf{S1_A}$ repeatedly until it finally destroys it.

Before, during or after the previous events[6], the following events occur:

- **E** sends a group of units ($\mathbf{U2_E}$) towards the eastern expansion site, the *natural*[7];
- **A** observes $\mathbf{U2_E}$ and sends a unit ($\mathbf{U2_A}$) to attack them;
- $\mathbf{U2_E}$ attacks $\mathbf{U2_A}$ and defeats it;
- $\mathbf{U2_E}$ keeps moving towards the natural and finally reaches it;
- $\mathbf{U2_E}$ starts building a command centre ($\mathbf{S2_E}$);
- **A** sends another group of units ($\mathbf{U3_A}$) to attack $\mathbf{U2_E}$;
- $\mathbf{U2_E}$ evades $\mathbf{U3_A}$, leading them away from the partially-built command centre;
- $\mathbf{U2_E}$ finishes building $\mathbf{S2_E}$ and stays in position near it;
- **E** starts recruiting units at the newly-built command centre;
- The group of units $\mathbf{U3_A}$ comes back, but this time $\mathbf{U2_E}$ attacks them back;
- $\mathbf{U3_A}$ moves away from $\mathbf{S2_E}$, but comes back after some time; and
- $\mathbf{U2_E}$ attacks $\mathbf{U3_A}$ once again to defend $\mathbf{S2_E}$.

Opponent's Rationale

The opponent's train of thought in this scenario is very straightforward. He is simply trying to achieve two main objectives while reacting to external and environmental events: (1) attack and destroy the observer's nearby base and stationed units ($\mathbf{S1_A}$); and (2) build a new command centre ($\mathbf{S2_E}$) to increase the unit production rate, and then defend the newly-built structure. A command centre is a very valuable structure for a player and it should thus be defended at all costs: destroying the observer's and building a new one would give the opponent a significant advantage in winning the game.

6. Events pertaining to $\mathbf{U1_E}$ and events pertaining to $\mathbf{U2_E}$ form two independent sub-scenarios that could be intertwined in a number of ways.

7. This expansion site is called the *natural* because it is closest to the opponent's home base ($\mathbf{S1_E}$), and thus a good, natural choice for a first expansion.

Observer's Rationale

The observer's actions are also very straightforward and do not require much further explanation. He is mainly reacting to the opponent's actions by attacking the encountered enemy units and trying to destroy the command centre ($\mathbf{S2}_E$) the opponent started (and later finished) building. In order to destroy $\mathbf{S2}_E$, the observer used a harassment tactic, much like the opponent did to destroy $\mathbf{S1}_A$ in the scenario. This tactic consists in repeatedly attacking and moving away before being attacked by the target: it was also used in the Cyclic Behaviour scenario in Section A.9.

Algorithmic Problem

The complexity in this scenario mainly comes from three factors: (1) the opponent is pursuing multiple objectives simultaneously; (2) the opponent is using cyclic behaviours; and (3) the sheer length of the scenario. Problems (1) and (2) have already been discussed previously (in Section A.7 and in Section A.9, respectively). The last problem, the length of the scenario, can be problematic for a number of reasons. For example, the algorithm should be able to understand that actions observed at different times during the scenario can be related to the same objective, even if the interval between the events is large. This includes not discarding actions that were observed a long time ago. Moreover, the number of hypotheses that explain the observed actions may grow along with the number of observations, which could potentially increase computation time and memory usage.

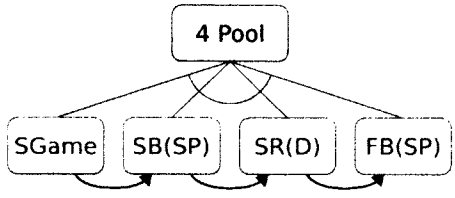
Annexe B

Bibliothèques de plans pour les expérimentations

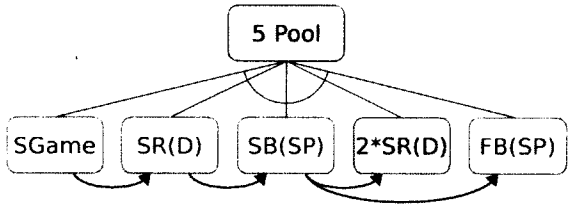
B.1 Stratégies d'ouverture dans le jeu *StarCraft : Brood War*

La figure B.1 ci-dessous donne des stratégies d'ouverture d'un joueur *Zerg* contre un joueur *Terran* dans le jeu *StarCraft : Brood War*, mieux connues dans le jargon des jeux de stratégie en temps réel sous le nom de *build orders*. Ces stratégies dictent l'ordre de construction des structures et de recrutement des unités en début de partie. Les noms de ces stratégies donnent le nombre d'unités ouvrières à recruter avant de construire une structure en particulier (par exemple, la stratégie *9 Pool* stipule que le joueur doit recruter 9 unités ouvrières, ou *Drones*, avant de construire un *Spawning Pool*, une structure permettant de recruter des unités de combat). Chaque stratégie a des avantages et des inconvénients : par exemple, la stratégie *4 Pool* permet de surprendre l'adversaire avec une attaque très tôt dans la partie, mais si cette attaque ne réussit pas à éliminer l'adversaire, le joueur aura très peu de chances de surmonter son désavantage économique par la suite.

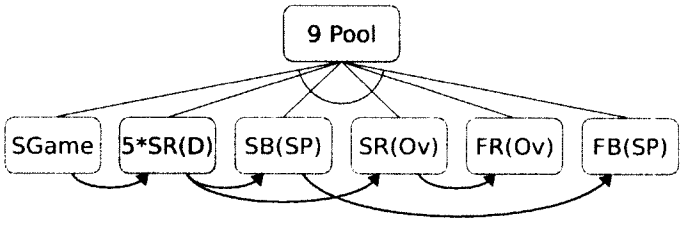
Ce sont des stratégies qui ont soigneusement été développées et éprouvées par les joueurs professionnels au fil du temps. Ainsi, pratiquement tous les joueurs de haut calibre suivent ces recettes à la lettre puisqu'elles sont optimales. La bibliothèque a



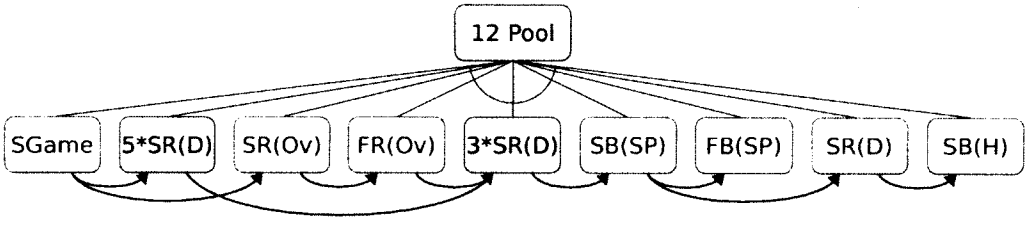
(a) *4 Pool*



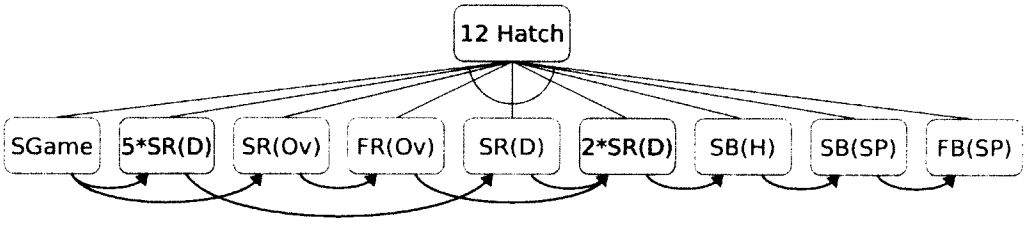
(b) *5 Pool*



(c) *9 Pool*



(d) *12 Pool*



(e) *12 Hatch*

Figure B.1 – Stratégies d'ouverture dans le jeu *StarCraft : Brood War*

été conçue à l'aide de la base de connaissances du site Wiki de la communauté *Team Liquid*[1].

L'action SGame (*Start Game*) a artificiellement été ajoutée à chaque plan pour signifier qu'une seule de ces stratégies peut évidemment être choisie par un joueur en début de partie et ainsi empêcher que l'algorithme ne génère des hypothèses expliquant les actions de l'agent par deux stratégies d'ouverture simultanées. La notation *Action(Paramètre)* est utilisée pour les actions : *n*Action(Paramètre)* représente un nœud « ET » sous lequel se trouvent n répétitions de l'action spécifiée, avec $n-1$ contraintes de précédence de façon à ordonner totalement les actions.

Voici la légende complète pour les actions composant cette bibliothèque de plans :

- **SGame** Start Game
- **SB** Start Building
- **FB** Finish Building
- **CB** Cancel Building
- **SR** Start Recruiting
- **FR** Finish Recruiting
- **SP** Spawning Pool
- **D** Drone
- **Ex** Extractor
- **Ov** Overlord
- **H** Hatchery

Le code XML pour cette bibliothèque de plans est donné dans l'encadré ci-dessous :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE PlanLibrary SYSTEM "htn.dtd">
<PlanLibrary>
 <!-- 12 Hatch -->
 <Plan probability="0.6">
  <AndTask name="12-hatch">
   <PrimitiveAction name="start-game" observation="Start Game"/>
   <AndTask name="5-workers">
    <PrimitiveAction name="start-worker-5" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-6" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-7" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-8" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-9" observation="Start Drone"/>
```

1. http://wiki.teamliquid.net/starcraft/

```
    <Constraint from="start-worker-5" to ="start-worker-6"/>
    <Constraint from="start-worker-6" to ="start-worker-7"/>
    <Constraint from="start-worker-7" to ="start-worker-8"/>
    <Constraint from="start-worker-8" to ="start-worker-9"/>
   </AndTask>
   <PrimitiveAction name="start-overlord-2" observation="Start Overlord"/>
   <PrimitiveAction name="finish-overlord-2" observation="Finish Overlord"/>
   <PrimitiveAction name="start-worker-10" observation="Start Drone"/>
   <AndTask name="2-workers">
    <PrimitiveAction name="start-worker-11" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-12" observation="Start Drone"/>
    <Constraint from="start-worker-11" to="start-worker-12"/>
   </AndTask>
   <PrimitiveAction name="start-hatch-2" observation="Start Hatchery"/>
   <PrimitiveAction name="start-pool" observation="Start Spawning Pool"/>
   <PrimitiveAction name="finish-pool" observation="Finish Spawning Pool"/>
   <Constraint from="start-game" to="5-workers"/>
   <Constraint from="start-game" to="start-overlord-2"/>
   <Constraint from="start-overlord-2" to="finish-overlord-2"/>
   <Constraint from="5-workers" to="start-worker-10"/>
   <Constraint from="start-worker-10" to="2-workers"/>
   <Constraint from="finish-overlord-2" to="2-workers"/>
   <Constraint from="2-workers" to="start-hatch-2"/>
   <Constraint from="start-hatch-2" to="start-pool"/>
   <Constraint from="start-pool" to="finish-pool"/>
  </AndTask>
</Plan>

<Plan probability="0.1">
 <AndTask name="12-pool">
  <PrimitiveAction name="start-game" observation="Start Game"/>
  <AndTask name="5-workers">
   <PrimitiveAction name="start-worker-5" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-6" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-7" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-8" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-9" observation="Start Drone"/>
   <Constraint from="start-worker-5" to ="start-worker-6"/>
   <Constraint from="start-worker-6" to ="start-worker-7"/>
   <Constraint from="start-worker-7" to ="start-worker-8"/>
   <Constraint from="start-worker-8" to ="start-worker-9"/>
  </AndTask>
  <PrimitiveAction name="start-overlord-2" observation="Start Overlord"/>
  <PrimitiveAction name="finish-overlord-2" observation="Finish Overlord"/>
  <AndTask name="3-workers">
   <PrimitiveAction name="start-worker-10" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-11" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-12" observation="Start Drone"/>
```

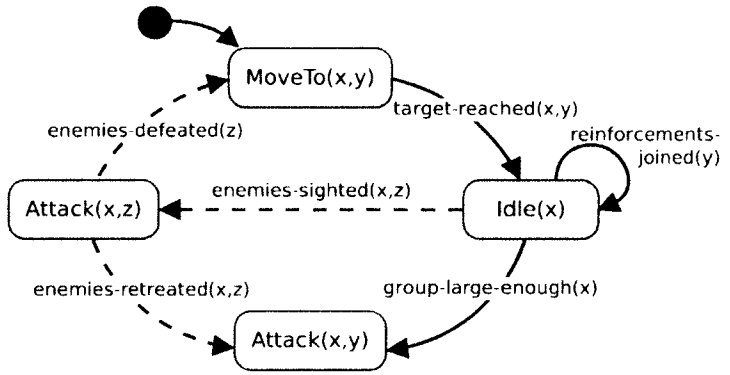
```
    <Constraint from="start-worker-10" to ="start-worker-11"/>
    <Constraint from="start-worker-11" to ="start-worker-12"/>
   </AndTask>
   <PrimitiveAction name="start-pool" observation="Start Spawning Pool"/>
   <PrimitiveAction name="finish-pool" observation="Finish Spawning Pool"/>
   <PrimitiveAction name="start-worker-13" observation="Start Drone"/>
   <PrimitiveAction name="start-hatch-2" observation="Start Hatchery"/>
   <Constraint from="start-game" to="5-workers"/>
   <Constraint from="start-game" to="start-overlord-2"/>
   <Constraint from="start-overlord-2" to="finish-overlord-2"/>
   <Constraint from="5-workers" to="3-workers"/>
   <Constraint from="finish-overlord-2" to="3-workers"/>
   <Constraint from="3-workers" to="start-pool"/>
   <Constraint from="start-pool" to="finish-pool"/>
   <Constraint from="start-pool" to="start-worker-13"/>
   <Constraint from="start-worker-13" to="start-hatch-2"/>
  </AndTask>
</Plan>

<Plan probability="0.2">
 <AndTask name="9-pool">
  <PrimitiveAction name="start-game" observation="Start Game"/>
  <AndTask name="5-workers">
   <PrimitiveAction name="start-worker-5" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-6" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-7" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-8" observation="Start Drone"/>
   <PrimitiveAction name="start-worker-9" observation="Start Drone"/>
   <Constraint from="start-worker-5" to ="start-worker-6"/>
   <Constraint from="start-worker-6" to ="start-worker-7"/>
   <Constraint from="start-worker-7" to ="start-worker-8"/>
   <Constraint from="start-worker-8" to ="start-worker-9"/>
  </AndTask>
  <PrimitiveAction name="start-pool" observation="Start Spawning Pool"/>
  <PrimitiveAction name="start-overlord-2" observation="Start Overlord"/>
  <PrimitiveAction name="finish-overlord-2" observation="Finish Overlord"/>
  <PrimitiveAction name="finish-pool" observation="Finish Spawning Pool"/>
  <Constraint from="start-game" to="5-workers"/>
  <Constraint from="5-workers" to="start-pool"/>
  <Constraint from="5-workers" to="start-overlord-2"/>
  <Constraint from="start-pool" to="finish-pool"/>
  <Constraint from="start-overlord-2" to="finish-overlord-2"/>
 </AndTask>
</Plan>

<Plan probability="0.05">
 <AndTask name="5-pool">
```

```
   <PrimitiveAction name="start-game" observation="Start Game"/>
   <PrimitiveAction name="start-worker-5" observation="Start Drone"/>
   <PrimitiveAction name="start-pool" observation="Start Spawning Pool"/>
   <AndTask name="2-workers">
    <PrimitiveAction name="start-worker-6" observation="Start Drone"/>
    <PrimitiveAction name="start-worker-7" observation="Start Drone"/>
    <Constraint from="start-worker-6" to ="start-worker-7"/>
   </AndTask>
   <PrimitiveAction name="finish-pool" observation="Finish Spawning Pool"/>
   <Constraint from="start-game" to="start-worker-5"/>
   <Constraint from="start-worker-5" to="start-pool"/>
   <Constraint from="start-pool" to="2-workers"/>
   <Constraint from="start-pool" to="finish-pool"/>
  </AndTask>
 </Plan>

 <Plan probability="0.05">
  <AndTask name="4-pool">
   <PrimitiveAction name="start-game" observation="Start Game"/>
   <PrimitiveAction name="start-pool" observation="Start Spawning Pool"/>
   <PrimitiveAction name="start-worker-4" observation="Start Drone"/>
   <PrimitiveAction name="finish-pool" observation="Finish Spawning Pool"/>
   <Constraint from="start-game" to="start-pool"/>
   <Constraint from="start-pool" to="start-worker-4"/>
   <Constraint from="start-worker-4" to="finish-pool"/>
  </AndTask>
 </Plan>
</PlanLibrary>
```

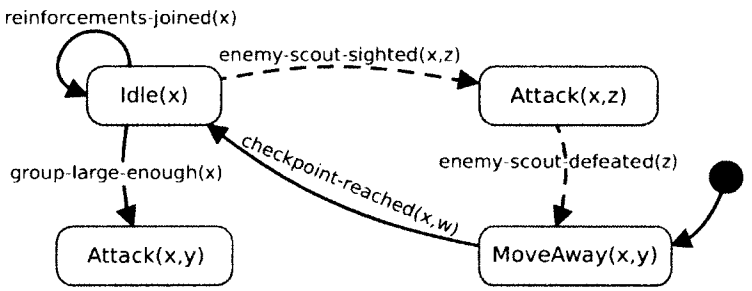
B.2 Stratégies tactiques dans un jeu RTS

La figure B.2 ci-dessous représente graphiquement des stratégies tactiques pouvant être exécutés par un agent dans un jeu de stratégie en temps réel, comme *StarCraft : Brood War*. Ces comportements, additionnés de celui de la figure 3.2, sont suffisants pour modéliser les scénarios de provocation des sections A.5 et A.6. Cependant, une bibliothèque de comportements plus étendue serait nécessaire pour reconnaître tous les comportements possibles d'un joueur durant une partie complète, ou même dans les autres scénarios de l'annexe A.

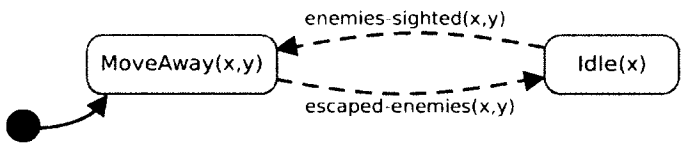
Le code XML pour cette bibliothèque de comportements (incluant le comportement de la figure 3.2) est donné dans l'encadré ci-dessous. Notons que les transitions



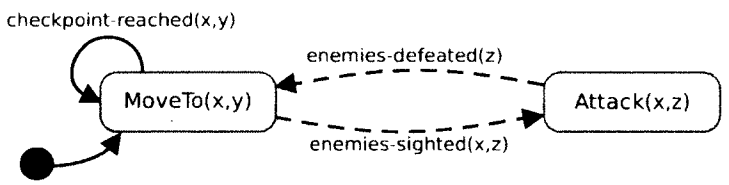
(a) Invasion – *Invade*(x, y)



(b) Embuscade – *Ambush*(x, y)



(c) Évitement – *AvoidEnemies*(x, y)



(d) Patrouille – *Patrol*(x, y)

Figure B.2 – Stratégies tactiques pour un jeu de stratégie en temps réel

sont toutes équiprobables et sont donc omises de la définition de la bibliothèque de plans.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BehaviourLibrary SYSTEM "fsm.dtd">
<BehaviourLibrary>
 <!-- Invade ?unit ?loc -->
 <Behaviour name="Invade(?unit,?loc)" probability="0.2" initial="move-to">
  <State id="move-to" action="MoveTo(?unit,?loc)">
   <Transition event="target-reached(?unit,?loc)" provokable="false" to="idle"/>
  </State>
  <State id="idle" action="Idle(?unit)">
   <Transition event="reinforcements-arrived(?loc)" provokable="false" to="idle"/>
   <Transition event="group-large-enough(?unit)" provokable="false" to="attack-b"/>
   <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="attack-u"/>
  </State>
  <State id="attack-u" action="Attack(?unit,?enemies)">
   <Transition event="enemies-defeated(?enemies)" provokable="true" to="move-to"/>
   <Transition event="enemies-retreated(?unit,?enemies)" provokable="true" to="attack-b"/>
  </State>
  <State id="attack-b" action="Attack(?unit,?loc)"/>
 </Behaviour>

 <!-- Contain ?unit ?loc -->
 <Behaviour name="Contain(?unit,?loc)" probability="0.2" initial="move-to">
  <State id="move-to" action="MoveTo(?unit,?loc)">
   <Transition event="target-reached(?unit,?loc)" provokable="false" to="idle"/>
  </State>
  <State id="idle" action="Idle(?unit)">
   <Transition event="reinforcements-joined(?unit)" provokable="false" to="idle"/>
   <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="attack-u"/>
  </State>
  <State id="attack-u" action="Attack(?unit,?enemies)">
   <Transition event="enemies-defeated(?enemies)" provokable="true" to="move-to"/>
   <Transition event="enemies-retreated(?unit,?enemies)" provokable="true" to="move-to"/>
  </State>
 </Behaviour>

 <!-- Ambush ?unit ?enemies -->
 <Behaviour name="Ambush(?unit,?enemies)" probability="0.2" initial="move-away">
  <State id="move-away" action="MoveAway(?unit,?enemies)">
   <Transition event="location-reached(?unit,?loc)" provokable="false" to="idle"/>
  </State>
  <State id="idle" action="Idle(?unit)">
   <Transition event="reinforcements-joined(?unit)" provokable="false" to="idle"/>
   <Transition event="group-large-enough(?unit)" provokable="false" to="attack-u"/>
   <Transition event="enemy-scout-sighted(?unit,?scout)" provokable="true" to="attack-s"/>
  </State>
  <State id="attack-s" action="Attack(?unit,?scout)">
```

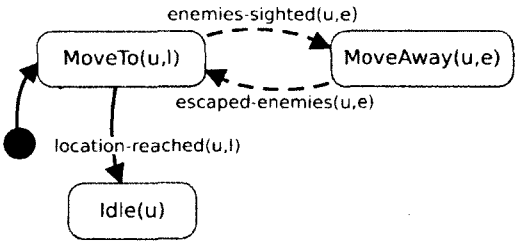
```
    <Transition event="enemy-scout-defeated(?scout)" provokable="true" to="move-away"/>
   </State>
   <State id="attack-u" action="Attack(?unit,?enemies)"/>
  </Behaviour>

  <!-- Avoid(?unit ... -->
  <Behaviour name="Avoid(?unit,?enemies)" probability="0.2" initial="move-away">
   <State id="move-away" action="MoveAway(?unit,?enemies)">
    <Transition event="escaped-enemies(?unit,?enemies)" provokable="true" to="idle"/>
   </State>
   <State id="idle" action="Idle(?unit)">
    <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="move-away"/>
   </State>
  </Behaviour>

  <!-- Patrol(?unit,?loc) ... -->
  <Behaviour name="Patrol(?unit,?loc)" probability="0.2" initial="move-to">
   <State id="move-to" action="MoveTo(?unit,?loc)">
    <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="attack-u"/>
    <Transition event="location-reached(?unit,?loc)" provokable="false" to="move-to"/>
   </State>
   <State id="attack-u" action="Attack(?unit,?enemies)">
    <Transition event="enemies-defeated(?enemies)" provokable="true" to="move-to"/>
   </State>
  </Behaviour>
</BehaviourLibrary>
```

B.3 Comportements tactiques dans un jeu RTS

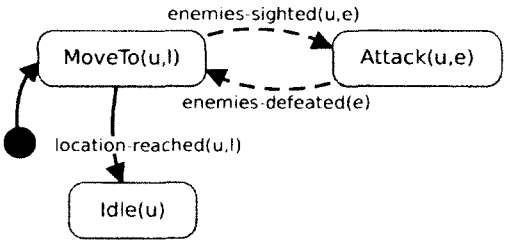
La figure B.3 ci-dessous illustre des comportements tactiques pouvant aussi être utilisés pour modéliser le comportement d'un agent dans un jeu de stratégie en temps réel. Ces comportements sont des actions de plus bas niveau d'abstraction que ceux dans la section précédente. Encore une fois, cette bibliothèque de comportements pourrait être bonifiée en y ajoutant davantage de comportements, permettant ainsi de reconnaître des scénarios plus complexes.

Le code XML pour cette bibliothèque de comportements est donné dans l'encadré ci-dessous. Notons que les transitions sont toutes équiprobables et sont donc omises de la définition de la bibliothèque de plans.

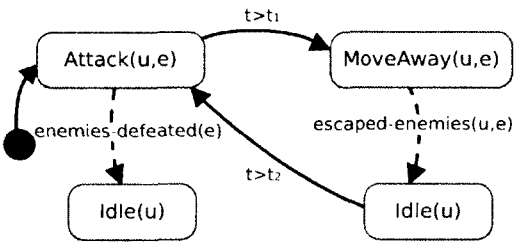
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BehaviourLibrary SYSTEM "fsm.dtd">
<BehaviourLibrary>
```



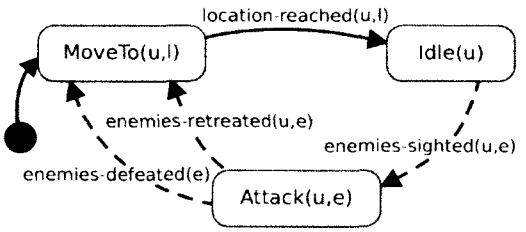
(a) Déplacement défensif – $Move(u, l)$



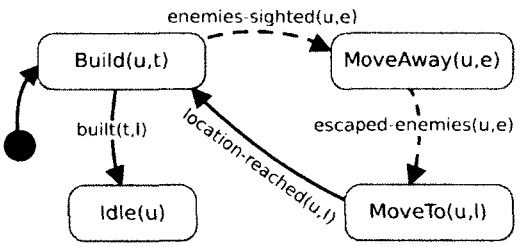
(b) Déplacement offensif – $AttackMove(u, l)$



(c) Harcèlement – $Harass(u, e)$



(d) Défense – $Defend(u, l)$



(e) Construction – $Build(u, t, l)$

Figure B.3 – Comportements tactiques pour un jeu de stratégie en temps réel

```
<!-- Move -->
<Behaviour name="Move(?unit,?loc)" probability="0.2" initial="move-to">
 <State id="move-to" action="MoveTo(?unit,?loc)">
  <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="move-away"/>
  <Transition event="location-reached(?unit,?loc)" provokable="false" to="idle"/>
 </State>
 <State id="move-away" action="MoveAway(?unit,?enemies)">
  <Transition event="escaped-enemies(?unit,?enemies)" provokable="true" to="move-to"/>
 </State>
 <State id="idle" action="Idle(?unit)"/>
</Behaviour>

<!-- AttackMove -->
<Behaviour name="AttackMove(?unit,?loc)" probability="0.2" initial="move-to">
 <State id="move-to" action="MoveTo(?unit,?loc)">
  <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="attack"/>
  <Transition event="location-reached(?unit,?loc)" provokable="false" to="idle"/>
 </State>
 <State id="attack" action="Attack(?unit,?enemies)">
  <Transition event="enemies-defeated(?enemies)" provokable="true" to="move-to"/>
 </State>
 <State id="idle" action="Idle(?unit)"/>
</Behaviour>

<!-- Harass -->
<Behaviour name="Harass(?unit,?enemies)" probability="0.2" initial="attack">
 <State id="attack" action="Attack(?unit,?enemies)">
  <Transition event="?t>t1" provokable="false" to="move-away"/>
  <Transition event="enemies-defeated(?enemies)" provokable="true" to="end"/>
 </State>
 <State id="move-away" action="MoveAway(?unit,?enemies)">
  <Transition event="escaped-enemies(?unit,?enemies)" provokable="true" to="wait"/>
 </State>
 <State id="wait" action="Idle(?unit)">
  <Transition event="?t>t2" provokable="false" to="attack"/>
 </State>
 <State id="end" action="Idle(?unit)"/>
</Behaviour>

<!-- Defend -->
<Behaviour name="Defend(?unit,?loc)" probability="0.2" initial="move-to">
 <State id="move-to" action="MoveTo(?unit,?loc)">
  <Transition event="location-reached(?unit,?loc)" provokable="false" to="idle"/>
 </State>
 <State id="idle" action="Idle(?unit)">
  <Transition event="enemies-sighted(?unit,?enemies)" provokable="true" to="attack"/>
 </State>
 <State id="attack" action="Attack(?unit,?enemies)">
  <Transition event="enemies-defeated(?enemies)" provokable="true" to="move-to"/>
```

Bibliographie

[1] Agence OTAN de normalisation, « NATO Glossary of Terms and Definitions / Glossaire OTAN de termes et définitions, » Organisation du traité de l'Atlantique Nord, Allied Administrative Publications (AAP-6), 2010.

[2] D. W. Albrecht, I. Zukerman, et A. E. Nicholson, « Bayesian Models for Keyhole Plan Recognition in an Adventure Game, » *User Modeling and User-Adapted Interaction*, vol. 8, pp. 5–47, janvier 1998.

[3] E. Alpaydin, *Introduction to Machine Learning*, 2e édition. Cambridge, MA: The MIT Press, 2010.

[4] D. Avrahami-Zilberbrand, « Efficient Hybrid Algorithms for Plan Recognition and Detection of Suspicious and Anomalous Behavior, » Thèse de doctorat, Bar-Ilan University, 2009.

[5] D. Avrahami-Zilberbrand et G. A. Kaminka, « Fast and Complete Symbolic Plan Recognition, » dans *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 653–658.

[6] D. Avrahami-Zilberbrand et G. A. Kaminka, « Towards Dynamic Tracking of Multi-Agents Teams: An Initial Report, » dans *Proceedings of the AAAI Workshop on Plan, Activity and Intent Recognition (PAIR)*, 2007, pp. 17–22.

[7] J. Azarewicz, G. Fala, R. Fink, et C. Heithecker, « Plan Recognition for Airborne Tactical Decision Making, » dans *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, 1986, pp. 805–811.

BIBLIOGRAPHIE

[8] F. Bacchus et F. Kabanza, « Using Temporal Logics to Express Search Control Knowledge for Planning, » *Artificial Intelligence*, vol. 116, no. 1-2, pp. 123-191, 2000.

[9] C. Baier et J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA: The MIT Press, 2008.

[10] C. L. Baker, R. Saxe, et J. B. Tenenbaum, « Action Understanding as Inverse Planning, » *Cognition*, vol. 113, pp. 329–349, 2009.

[11] S. Bakkes, P. Spronck, et J. van den Herik, « Opponent Modelling for Case-based Adaptive Game AI, » *Entertainment Computing*, vol. 1, no. 1, pp. 27–37, 2009.

[12] B. Banerjee et L. Kraemer, « Branch and Price for Multi-Agent Plan Recognition, » dans *Proceedings of the 25th Conference on Artificial Intelligence (AAAI)*, 2011, pp. 601–607.

[13] A. R. Benaskeur, F. Kabanza, et E. Beaudry, « CORALS: A Real-Time Planner for Anti-Air Defence Operations, » *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 2, pp. 1–21, 2010.

[14] D. S. Bernstein, R. Givan, N. Immerman, et S. Zilberstein, « The Complexity of Decentralized Control of Markov Decision Processes, » *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819-840, 2002.

[15] F. Bisson et F. Kabanza, « Eliciting Plan Recognition Cues by Provoking Opponents in RTS Games, » dans *Dagstuhl Seminar on Plan Recognition*, avril 2011.

[16] F. Bisson, F. Kabanza, et P. Bellefeuille, « La reconnaissance de plan des adversaires, » dans *Programme du 79e Congrès de l'Association francophone pour le savoir (Acfas)*, mai 2011.

[17] F. Bisson, F. Kabanza, A. R. Benaskeur, et H. Irandoust, « Provoking Opponents to Facilitate the Recognition of their Intentions, » dans *Proceedings of the AAAI Student Abstract and Poster Program*, 2011, pp. 1762–1763.

BIBLIOGRAPHIE

[18] N. Blaylock. « Thinking about Evaluation and Corpora for Plan Recognition. » dans *Dagstuhl Seminar on Plan Recognition*. 2011.

[19] A. L. Blum et M. L. Furst. « Fast Planning Through Planning Graph Analysis. » *Artificial Intelligence*, vol. 90, no. 1–2, pp. 281–300, février 1997.

[20] B. Bouchard. « Un modèle de reconnaissance de plans pour les personnes atteintes de la maladie d'Alzheimer basé sur la théorie des treillis et sur un modèle d'actions en logique de description. » Thèse de doctorat, Université de Sherbrooke, 2006.

[21] M. Buro. « Real-Time Strategy Games: A new AI Research Challenge. » dans *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 1534–1535.

[22] S. Carberry, *Plan Recognition in Natural Language Dialogue*, série ACL-MIT Press Series on Natural Language Processing. Cambridge, MA: The MIT Press, 1990.

[23] S. Chamberland et F. Kabanza, « Heuristic Planning in Adversarial Dynamic Domains, » dans *Proceedings of the AAAI Student Abstract and Poster Program*, 2011, pp. 1766–1767.

[24] E. Charniak et R. P. Goldman. « A Probabilistic Model of Plan Recognition. » dans *Prooceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, 1991, pp. 160–165.

[25] E. Charniak et D. V. McDermott, *Introduction to Artificial Intelligence*. Reading, MA: Addison-Wesley, 1985.

[26] P. R. Cohen, C. R. Perrault, et J. F. Allen. « Beyond Question Answering. » dans *Strategies for Natural Language Processing*. W. G. Lehnert et M. H. Ringle, éditeurs. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981, pp. 245–274.

[27] T. M. Cover et J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[28] G. De Giacomo, F. Patrizi, et S. Sardina, « Agent Programming via Planning Programs. » dans *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010, pp. 491–498.

[29] C. Elsaesser et F. J. Stech, « Detecting Deception. » dans *Adversarial Reasoning: Computational Approaches to Reading The Opponent's Mind.* Chapman & Hall/CRC, 2006, pp. 101–124.

[30] D. Fox, W. Burgard, et S. Thrun, « Active Markov Localization for Mobile Robots. » *Robotics and Autonomous Systems*, vol. 25, no. 3–4, pp. 195–207, 1998.

[31] A. Frini et A. R. Benaskeur, « INCOMMANDS TDP: Measures of Performance for the Threat Evaluation Capability. » Defence R&D Canada – Valcartier, Rapport technique 2009-240, 2009.

[32] G. J. Funke et S. M. Galster, « The Effects of Cognitive Processing Load and Collaboration Technology on Team Performance in a Simulated Command and Control Environment, » *International Journal of Industrial Ergonomics*, vol. 39, no. 3, pp. 541–547, 2009.

[33] C. W. Geib, « Assessing the Complexity of Plan Recognition. » dans *Proceedings of the 19th Conference on Artificial Intelligence (AAAI)*, 2004, pp. 507–512.

[34] C. W. Geib et R. P. Goldman, « Plan Recognition in Intrusion Detection Systems. » dans *Proceedings of the DARPA Information Survivability Conference & Exposition II (DISCEX-II)*, 2001, pp. 46–55.

[35] C. W. Geib et R. P. Goldman, « Partial Observability and Probabilistic Plan/Goal Recognition. » dans *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO)*, 2005.

[36] C. W. Geib et R. P. Goldman, « A Probabilistic plan recognition algorithm based on plan tree grammars. » *Artificial Intelligence*, vol. 117, no. 11, pp. 1101–1132, 2009.

[37] C. W. Geib et R. P. Goldman, « Recognizing Plans with Loops Represented in a Lexicalized Grammar, » dans *Proceedings of the 25th Conference on Artificial Intelligence (AAAI)*, 2011, pp. 958–963.

[38] C. W. Geib, J. Maraist, et R. P. Goldman, « A New Probabilistic Plan Recognition Algorithm Based on String Rewriting, » dans *International Conference on Automated Planning and Scheduling*, 2008, pp. 91–98.

[39] M. Ghallab, D. Nau, et P. Traverso, *Automated Planning: Theory and Practice*, San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[40] P. Gmytrasiewicz et P. Doshi, « A Framework for Sequential Planning in Multiagent Settings, » *Journal of Artificial Intelligence Research (JAIR)*, vol. 24, pp. 49–79, 2005.

[41] C. V. Goldman et S. Zilberstein, « Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis, » *Journal of Artificial Intelligence Research (JAIR)*, vol. 22, pp. 143–174, 2004.

[42] R. P. Goldman, C. W. Geib, et C. A. Miller, « A New Model of Plan Recognition, » dans *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 1999, pp. 245–254.

[43] H. Irandoust, A. R. Benaskeur, K. Baker, et S. Banbury, « Naval Force-Level Tactical Command and Control – Mission Analysis and Problem Characterization, » Defence R&D Canada – Valcartier, Rapport technique 2009-199, 2009.

[44] H. Irandoust, A. R. Benaskeur, F. Kabanza, et P. Bellefeuille, « A Mixed-Initiative Advisory System for Threat Evaluation, » dans *15th International Command and Control Research and Technology Symposium (ICCRTS)*, 2010.

[45] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, et H. Irandoust, « Opponent Behaviour Recognition for Real-Time Strategy Games, » dans *Proceedings of the AAAI Workshop on Plan, Activity and Intent Recognition (PAIR)*, 2010, pp. 29–36.

BIBLIOGRAPHIE

[46] H. A. Kautz et J. F. Allen. « Generalized Plan Recognition. » dans *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*. 1986. pp. 32–37.

[47] D. Keaveney et C. O'Riordan. « An Abstract Model for Real-Time Strategy Games. » National University of Ireland, Galway. Rapport technique. 2008.

[48] D. L. Kovács. « BNF Definition of PDDL 3.1. » 2011. 7th International Planning Competition.

[49] M. Lanctot, K. Waugh, M. Zinkevich, et M. Bowling. « Monte Carlo Sampling for Regret Minimization in Extensive Games. » *Advances in Neural Information Processing Systems 22 (NIPS)*. pp. 1078–1086. 2009.

[50] J. R. Landis et G. G. Koch. « The Measurement of Observer Agreement for Categorical Data. » *Biometrics*, vol. 33, pp. 150–174. 1977.

[51] K. Laviers, G. Sukthankar, M. Molineaux, et D. Aha. « Exploiting Early Intent Recognition for Competitive Advantage. » dans *Proceedings of the IJCAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*. 2009. pp. 58–63.

[52] N. Lesh. « Scalable and Adaptive Goal Recognition. » Thèse de doctorat. University of Washington. 1998.

[53] N. Lesh, C. Rich, et C. L. Sidner. « Using Plan Recognition in Human-Computer Collaboration. » dans *Proceedings of the 7th International Conference on User Modeling*. 1999. pp. 23–32.

[54] J. McCarthy. « Circumscription—A Form of Nonmonotonic Reasoning. » *Artificial Intelligence*. vol. 13. no. 1–2. pp. 27–39. 1980.

[55] F. Mulder et F. Voorbraak. « A Formal Description of Tactical Plan Recognition. » *Information Fusion*. vol. 4. no. 1. pp. 47–61. 2003.

[56] R. B. Myerson. *Game Theory: Analysis of Conflict*. Cambridge, MA: Harvard University Press. 1991.

[57] A. Newell. « The Knowledge Level. » *Artificial Intelligence*. vol. 18. no. 1. pp. 87–127. 1982.

[58] S. Paradis, A. R. Benaskeur, M. Oxenham, et P. Cutler, « Threat Evaluation and Weapons Allocation in Network-Centric Warfare, » dans *Proceedings of the 8th International Conference on Information Fusion*, 2005, pp. 1078–1085.

[59] A. Paz, *Introduction to Probabilistic Automata*. Orlando, FL: Academic Press, Inc., 1971.

[60] M. J. V. Ponsen, S. Lee-Urban, H. Muñoz Avila, D. W. Aha, et M. Molineaux, « Stratagus: An Open-Source Game Engine for Research in Real-Time Strategy Games, » dans *Proceedings of the 2005 IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005, pp. 78–83.

[61] D. V. Pynadath et M. P. Wellman, « Accounting for Context in Plan Recognition with Application to Traffic Monitoring, » dans *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1995, pp. 472–481.

[62] S. Rabin, *AI Game Programming Wisdom*. Rockland, MA: Charles River Media, Inc., 2002.

[63] M. Ramírez et H. Geffner, « Plan Recognition as Planning, » dans *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 1778–1783.

[64] M. Ramírez et H. Geffner, « Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners, » dans *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*, 2010, pp. 1121–1126.

[65] M. Ramírez et H. Geffner, « Goal Recognition over POMDPs: Inferring the Intention of a POMDP Agent, » dans *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, 2011, pp. 2009–2014.

[66] P. C. Roy, « Modèle possibiliste pour la reconnaissance d'activités: Habitat intelligent, » Thèse de doctorat, Université de Sherbrooke, 2011.

[67] S. Russell et P. Norvig, *Artificial Intelligence: A Mordern Approach*, 3e édition, Prentice Hall, 2010.

[68] A. Sadilek et H. Kautz, « Recognizing Multi-Agent Activities from GPS Data, » dans *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*, 2010, pp. 1134–1139.

[69] F. Sailer, M. Buro, et M. Lanctot, « Adversarial Planning Through Strategy Simulation, » dans *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 2007, pp. 80–87.

[70] F. Schadd, S. Bakkes, et P. Spronck, « Opponent Modelling in Real-Time Strategy Games, » dans *Proceedings of the 8th Conference on Intelligent Games and Simulation (GAME-ON)*, 2007, pp. 61–68.

[71] C. F. Schmidt, N. S. Sridharan, et J. L. Goodson, « The Plan Recognition Problem : an Intersection of Psychology and Artificial Intelligence, » *Artificial Intelligence*, vol. 11, pp. 45–83, 1978.

[72] C. F. Schmidt, « Understanding Human Action: Recognizing the Plans and Motives of Other Persons, » dans *Cognition and Social Behavior*, série Carnegie-Mellon University Cognition, J. S. Carroll et J. W. Payne, éditeurs. Lawrence Erlbaum Associates, 1976, pp. 47–68.

[73] O. C. Schrempf, U. D. Hanebeck, A. J. Schmid, et H. Wörn, « A Novel Approach to Proactive Human-Robot Cooperation, » dans *Proceedings of the IEEE International Workshop on Robots and Human Interactive Communication*, 2005, pp. 555–560.

[74] A. N. Steinberg, « Predictive Modeling of Interacting Agents, » dans *Proceedings of the 10th International Conference on Information Fusion*, 2007, pp. 1–6.

[75] A. N. Steinberg, « Foundations of Situation and Threat Assessment, » dans *Handbook of Multisensor Data Fusion: Theory and Practice*, série The Electrical Engineering and Applied Signal Processing, M. E. Liggins, D. L. Hall, et J. Llinas, éditeurs. CRC Press, Taylor & Francis Group, 2009, pp. 437–501.

[76] R. J. Sternberg, éditeur, *Thinking and Problem Solving*, série Handbook of Perception and Cognition. Academic Press, 1994.

BIBLIOGRAPHIE

[77] G. Sukthankar et K. Sycara, « A Cost Minimization Approach to Human Behavior Recognition, » dans *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005, pp. 1067–1074.

[78] G. Sukthankar et K. Sycara, « Simultaneous Team Assignment and Behavior Recognition from Spatio-temporal Agent Traces, » dans *Proceedings of the 21st Conference on Artifcial Intelligence (AAAI)*, 2006, pp. 716–721.

[79] G. Sukthankar et K. Sycara, « Hypothesis Pruning and Ranking for Large Plan Recognition Problems, » dans *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, 2008, pp. 998–1003.

[80] M. Tambe, « Towards Flexible Teamwork, » *Journal of Artificial Intelligence Research*, vol. 7, pp. 83–124, 1997.

[81] M. Tambe et P. S. Rosenbloom, « RESC: An Approach for Real-time, Dynamic Agent Tracking, » dans *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 1995, pp. 103–110.

[82] A. M. Turing, « Computing Machinery and Intelligence, » *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[83] US Department of Defense, « Doctrine for the Armed Forces of the United States, » Department of Defense, Joint Publication 1, 2009.

[84] R. Wilensky, *Planning and Understanding: A Computational Approach to Human Reasoning.* Reading, MA: Addison-Wesley, 1983.