Raport de Projet

Fondements des Systèmes Multiagents

Dans le cadre du
MASTER de Sorbonne Université

Spécialité
Informatique

Parcours
ANDROIDE

ZUO Nicolas — Étudiant (Gr1)
CALIC Petar — Étudiant (Gr1)
HERPSON Cédric — Encadrant
BEYNIER Aurélie — Encadrante
MAUDET Nicolas — Encadrant

Table des Matières

1 Introduction

Tout au long du semestre nous avons travaillé sur le projet dédale dans lequel il s'agit de développer une variante multi-agent d'un jeu inspiré de "Hunt the Wumpus " (Gregory Yobe, 1972). Dans la version d'origine, il s'agit pour le joueur d'aller tuer un monstre, le Golem, caché dans une caverne tout en ramassant des trésors. Dans notre cas, nous ne prenons pas en compte la partie chasse au trésor et nous devons bloquer le Golem. Nous allons donc implémenter, un Système Multi Agent dans lequel des agents dans un premier temps font une exploration de l'environnement puis dans un second temps la chasse au Golem.
L'environnement est représenté par un graphe où les nœuds sont les positions possibles et les arcs les passages. Le but étant donc de bloquer le Golem sur la carte, nous avons développé différentes stratégies exécuter par les agents afin d'assurer une bonne coopération et ainsi atteindre efficacement leur objectif. Nous utiliserons le langage de programmation Java et la plateforme multi-agent JADE. Les paramètres avec lesquels nous allons ensuite expérimenter sont : L'échelle de l'expérience (taille de la carte, nombre d'agents et de Golem) le radius de vision des agents et de l'odeur du Golem, la vitesse des agents un par rapport aux autres.

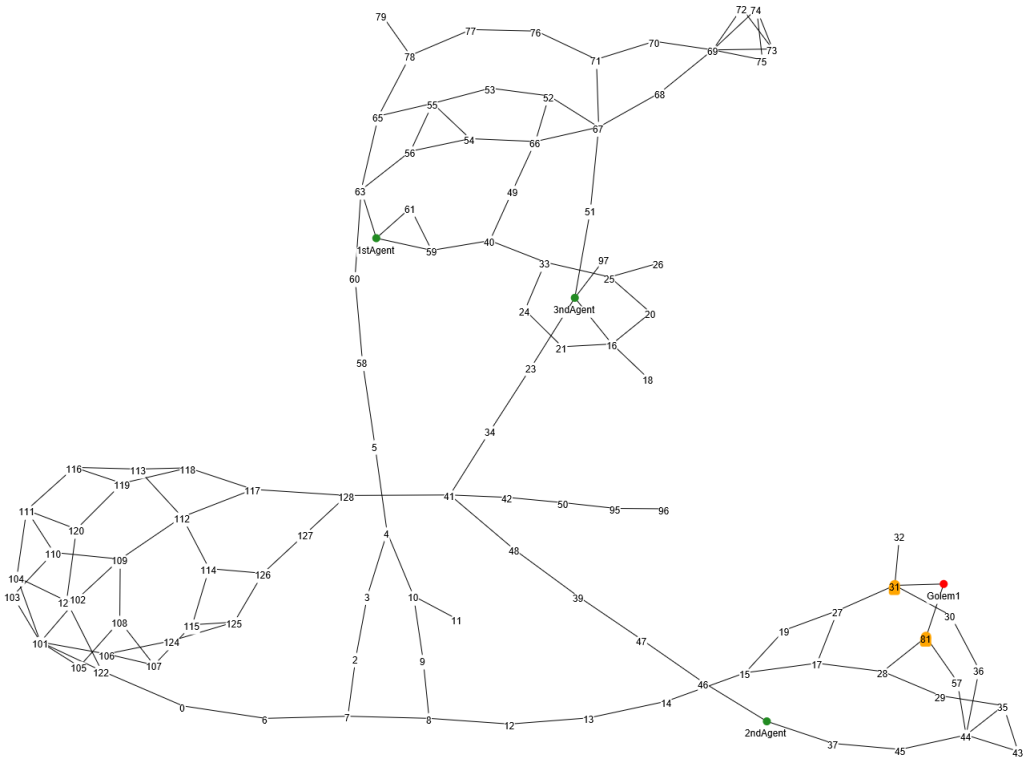

FIGURE 1 – Fenêtre de notre programme

2 Présentation des choix associés

Nous avons choisi de suivre un schéma de comportement avancé, composé, le FSM (Finite State Machine). Pour pouvoir contrôler la séquentialité entre les différents comportements d'un agent et de faire des transitions entre les Behaviours de façon automatisée, grâce à la plateforme Jade, qui nous permet de directement aller vers un comportement suivant dès que celui-ci est terminé.
Notre FSM comporte 13 comportements différents, dont trois principales, où se trouve la grande majorité du code, qui est l'**ExploCoop** pour la partie exploration, **DumbChase** pour la partie chasse du Wumpus et **MoveTo** un comportement de coordination pour arriver à une case précise le plus rapidement possible. Le reste des comportements sert à une bonne communication entre ces trois classes de comportements principalement.

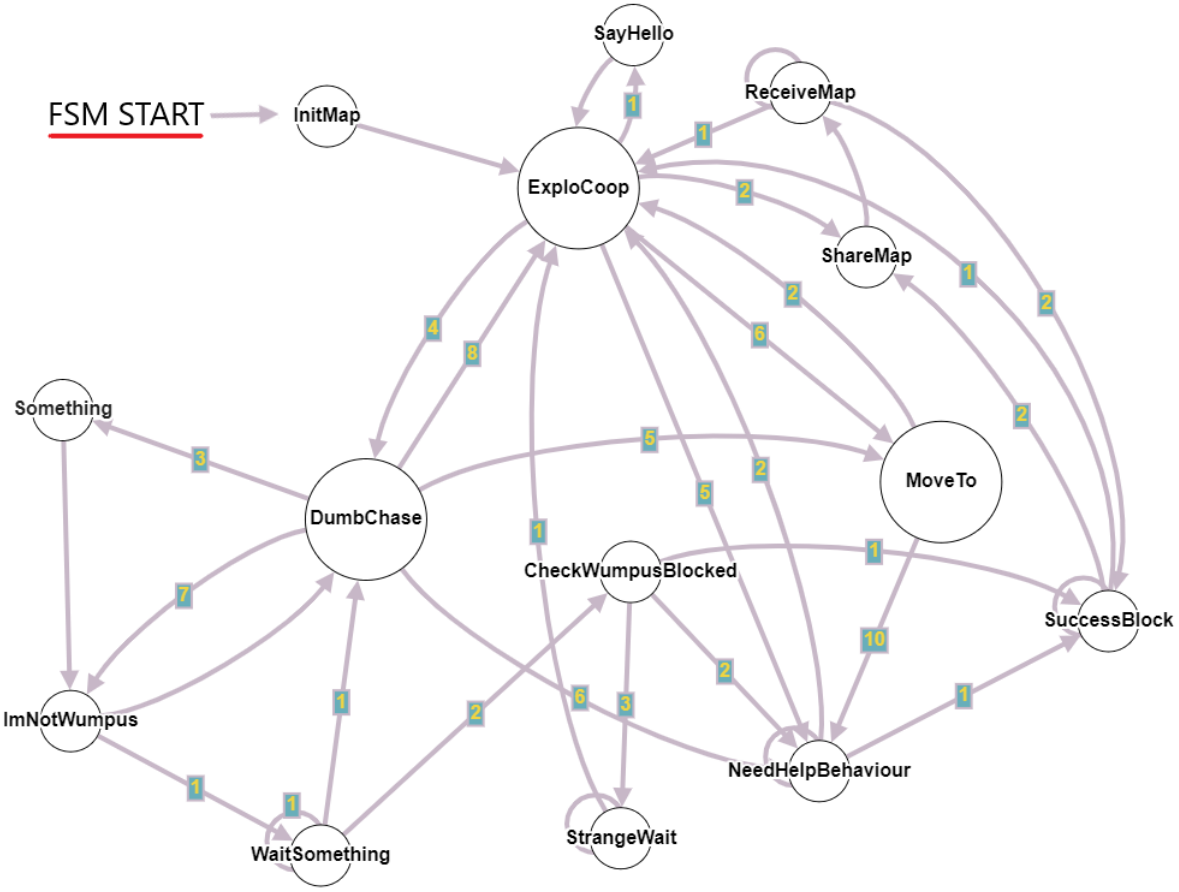

FIGURE 2 – Graphe des connections entre les comportements

2.1 Exploration

2.1.1 Le Principe

Notre exploration se comporte de manière à explorer en priorité en profondeur, puis en largeur. L'agent va donc explorer les nœuds ouverts autour de lui en priorité et s'il ne trouve plus de nœuds ouvert autour de lui, il va prendre le nœud ouvert le plus proche de lui possible. Mais ajouté à cela, il y a quelques exceptions à cette règle, quand l'agent vient de réussir à faire un échange réussit de carte avec un notre agent, alors un des deux agents est choisis pour générer un autre chemin, que celle de l'exploration normale, car les deux auront exactement la même carte et les mêmes nœuds ouverts a explorés, ils ont alors de grandes chances d'explorer toute la carte côte à côte, ce qui n'est pas très judicieux et souhaité. D'autres exceptions peuvent intervenir, car au cours des échanges, un agent peut rencontrer un autre agent qui lui dit de le contourner pour des raisons de potentiel blocage d'un Wumpus, l'agent en question génère alors des chemins en enlevant les nœuds où un autre agent lui a communiqué qu'il est inaccessible. Il y a aussi le fait que quand notre agent rencontre un agent avec un comportement **MoveTo**, il essaye de trouver une case autour de lui qui n'est pas sur le chemin de l'agent en mission pour lui laisser passer. Et une dernière chose, dans le comportement d'**ExploCoop**, a chaque itération il regarde dans sa boite au lettre tout type de message qu'il a reçu et il se comporte en fonction, par exemple un message avec un protocole de partage de carte, il va alors quitter le comportement d'exploration pour aller dans le comportement **ShareMap**, ou encore quand il reçoit un message avec un protocole d'aide de bloquer un potentiel Wumpus, il peut réagir en fonction, de s'il doit l'aider ou non, mais on détaillera cela dans la partie communication. Il y a aussi un certain niveau de sensibilité, où au bout d'un certain nombre d'itérations où notre agent n'a pas pu aller sur la case souhaitée, qui est bloqué par un autre agent ou Wumpus de ne pas directement lancer le protocole de vérification si c'est un agent ou un Wumpus. Pendant l'exploration, si notre agent bloque par hasard un Wumpus et qu'il est bloqué depuis longtemps, alors il reste sur place et indique aux autres agents qu'il a réussi a bloquer un Wumpus et de passer par un autre chemin si les autres agents veulent aller sur ca case. Et à chaque X temps l'agent s'arrête pour envoyer un message autour de lui pour savoir s'il y a un agent à côté pour s'échanger des informations comme l'échange de carte.

2.1.2 L'Avantage

L'avantage de ce système, c'est qu'il y a une centralisation d'un comportement principal, qui est l'exploration de la carte de manière a ce qu'il soit le plus rapide possible, et de pouvoir gérer à tout moment un message reçu d'un autre agent et de directement le traité avant de faire le prochain mouvement de case. Et ignorer les messages qui ne nous intéressent pas et qui ne nous concernent pas, pour se concentrer dans un premier dans un optique d'explorer le plus rapidement possible, pour après passer a la chasse.

2.1.3 Limites, Complexité, Optimalité et Critère d'Arrêt

Les limites de ce système est le fait qu'on se retrouve avec un comportement très lourd en termes de code, ce qui n'est pas optimal et souhaité, car il est préférable d'avoir des comportements légers, mais simple, et de les appeler au bon moment, car dans notre cas, chaque itération vérifie plusieurs fois ca boite aux lettres pour vérifier que chaque type de protocole de messages soit présent ou non, ce qui en termes de complexité n'est pas très efficace, mais néanmoins dans notre cas, il reste toute fois très correcte, car ce ne sont que des vérifications d'une liste de boite aux lettres. Pour la génération du chemin, on utilise l'algorithme de Dijkstra, qui est le plus optimal en termes de complexité pour la recherche d'un plus court chemin possible. Et aussi dépendant de bons paramètres initiaux qu'on initialise à nos agents, si nos agents sont trop rapides et que la sensibilité de nos agents avant de faire une vérification pour savoir si c'est un Golem ou non n'est pas assez élever, alors on peut perdre grandement en termes de vitesse d'exploration, car nos agents n'auront pas le temps de faire leur partage de carte, qu'ils seront en permanence en train de vérifier si les éléments qui lui bloquent sont un Wumpus ou non. Et l'inverse aussi pose des problèmes, si la sensibilité est trop élevée alors nos agents seront bloqué sur place trop longtemps avant de lancer une vérification, ce qui en terme d'optimalité n'est pas très efficace.
Pour la partie d'après que notre agent ait généré un chemin différent que d'habitude après un échange de carte réussit, l'algorithme n'est pas optimal et n'a pas pu être amélioré, par manque de temps, car dans notre cas le chemin généré pour se séparer de l'agent avec qui on vient de communiquer, génère sur un nœud au hasard sur ca connaissance de la carte, ce qui a des moments fait que notre agent continu d'être collé avec un autre agents en exploration, mais la partie exploration est assez robuste et ne comporte pas d'incohérences ou de comportements non voulu à la base. Pour les critères d'arrêt de ce comportement il y en a plusieurs, quand il reçoit un message de protocole en particulier comme d'aider a bloquer un Wumpus, partage de carte, quand pendant l'exploration il trouve par hasard un Wumpus où il l'a bloqué par chance, ou encore a chaque X temps l'agent s'arrête pour faire un **SayHello** autour de lui, pour échanger avec d'autre agents, et pour finir quand l'exploration est fini, c'est-à-dire quand il n'y a plus aucun nœud ouvert, alors automatiquement il sera transféré dans le rôle de chasse.

2.2 Communication et Coordination

2.2.1 Le Principe

Il existe deux types de communication, l'un par envoi répétitif qui est surtout utilisé quand un agent a besoin d'échanger une information qu'il possède, et une autre qui est les communications par réaction et qui est ciblée ou non en fonction du contexte. Le fonctionnement de tous les comportements, principal (**ExploCoop**, **DumbChase** et **MoveTo**) mais aussi de quelques une en particulier comme **NeedHelpBehaviour**, fonctionne sur le principe de regarder sa boite au lettre avant de faire une action pour avoir toutes les connaissances, avant de faire une action, sauf pour MoveTo, où il n'accède au boite au lettre qu'une fois qu'il ait eu un échec de

déplacement vers une case, alors il regarde sa boite au lettre pour savoir ce qu'il se passe autour de lui et si ce n'est pas des informations importantes, il les ignore et envoie un message pour lui laisser passer, car il est prioritaire.
Donc les principaux comportements peuvent envoyer des messages tous les X temps, et quand il reçoit un message nécessitant un échange avec un agent, il renvoie un message ciblé à seulement l'agent qui lui envoyait le message. Un comportement peut faire le choix de directement faire le traitement d'envoi du message dans le même comportement quand cela n'est pas long ou avec des complexités non impactantes, ou de laisser sa place à de nouveaux comportements qu'il fait diriger avant de s'arrêter, comme par exemple :

— Pour le partage de carte : tout les X temps **ExploCoop** laisse au comportement **SayHello** prendre la place pour envoyer des messages autour pour vérifier s'il y a des agents autours et il revient directement dans le comportement d'exploration, mais si cette fois l'agent reçoit un message d'un autre agent de **SayHello**, il laisse sa place à **ShareMap** qui envoie sa carte pour le partager, et de directement passé après dans **ReceiveMap**, qui a le choix de se bloquer s'il n'a pas encore reçu le protocole de partage de carte de l'agent communiquant avec, mais si au bout d'un certain temps il ne reçoit toujours rien, il se réveille automatiquement et retourne a l'exploration quoi qu'il arrive. (à noter qu'une fois passer en mode chasse, l'agent en question peut encore envoyer des communications de partage de carte pour que les autres agents aussi complètes leur carte au complet, et il stocke en mémoire chaque agent qu'il a déjà envoyé pour ne plus lui renvoyer)

— Pour la vérification d'une case où on est quasiment sûr d'avoir un Wumpus a cette case, grâce à **WaitSomething**, puis on lance le protocole de vérification de la case et des comportements à suivre en fonction du type de case où se trouve le Wumpus bloquer dans **CheckWumpusBlocked**, ce comportement contient trois sorties possibles, mais dans tout les cas il regarde d'abord la case où il veut aller, si c'est une case avec un nœud ouvert, alors notre agent n'a aucune garantie qu'il a réellement bloqué le Wumpus, car rien ne nous dit que le Wumpus ne bouge pas volontairement et qu'il a encore des nœuds pour s'enfuir, il sera alors dirigé vers le comportement **StrangeWait**, qui communiquera autour de lui qu'il a probablement bloqué un Wumpus et de le contourner si possible, mais tout les X temps notre agent tentera d'accéder à ce nœud ouvert, pour continuer a vérifié si l'agent est encore bloqué ou non. Si on revient à la vérification, on se retrouve alors une autre possibilité qui est que le nœud soit fermer, si le nœud est fermé l'agent regarde autour de ce nœud qu'il pense que ça soit un Wumpus, s'il voit qu'il n'y a aucune sortie possible depuis le nœud fermé où se trouve le Wumpus alors il laisse la place au comportement **SuccessBlock** qui notifie aux alentours qu'il a bloqué l'agent avec succès et continue de partager sa carte **une fois avec tous les agents** qui l'envoie un message pour aider les agents en exploration. Et le dernier cas, quand le

nœud est fermé est qu'il a des nœuds autour du golem qui pourrait lui laisser s'enfuir, alors on passe dans le comportement **NeedHelpBehaviour**, qui est un comportement qui traite beaucoup d'information, mais qui envoie surtout des messages de demande d'aide a bloqué le Wumpus à l'agent autour de lui, il envoie alors un message sérialiser, contenant sa position, le nœud où se trouve le golem, et tous les nœuds où il nécessite blocage, et les autres agents qui sont en exploration ou en chasse, qui reçois le message du protocole d'aide a bloqué, on plusieurs réaction possible soit ils sont déjà sur une case à bloquer alors il fait une mise à jour de la liste des nœuds à bloqur et communique à l'agent autour, sinon ils laissent leur place au comportement MoveTo, qui est le comportement pour y accédé rapidement au nœud qu'il souhaite aller avec une priorité des mouvements, et dans ce comportement, il peut y avoir plusieurs cas possibles, soit il n'a pas réussi à accéder au nœud souhaité, alors il retourne à l'exploration (À noter que si l'exploration était déjà finie, alors le comportement **ExploCoop** redirigera automatiquement vers la chasse), mais s'il réussit à accéder au nœud souhaité, alors il arrive lui aussi dans le comportement **NeedHelpBehaviour** et dans cet état, il y a plusieurs possibilités soit c'était une fausse alerte ou que le Wumpus à bouger entre temps ou en fin de compte, c'est pas un Wumpus, alors il retourne à l'exploration(ou chasse), soit il reste des nœuds a bloqué alors il continue a envoyé des aides de blocage, et si tous les nœuds sont bloqués alors il accède a l'état **SuccessBlock** !

— Pour la vérification rapide d'une case où l'on n'est pas du tout sûr si c'est un agent ou un Wumpus, est une suite de comportements, où le principe est qu'il fasse une vérification rapide avant de lancer **CheckWumpusBlocked** qui est un comportement où on est quasiment sûr que ça soit un Wumpus sur la case. Le déroulement de cette vérification rapide se comporte par trois états très simples **Something** qui est activé quand notre agent n'a pas pu accéder à une casse depuis une certaine sensibilité définies au départ, le comportement est très simple il envoie autour de lui avec comme contenu dans le message la position de la case qu'il veut accéder, puis passe automatiquement dans **ImNotWumpus** qui envoie autour de lui qu'il n'est pas un Wumpus, avec comme contenu la case actuel où il se trouve et qui retourne a la chasse s'il est précédemment venu de la chasse sinon il part automatiquement après dans **WaitSomething** qui attend un certain temps et s'il n'y a pas de réponse alors il considère que c'est un Wumpus et accède à **CheckWumpusBlocked** pour faire toutes les étapes suivantes, mais si c'est une fausse alerte alors il retourne a la chasse (à noter que l'agent qui reçoit un message Something, vérifie que le nœud que l'agent qu'il lui a envoyé le message correspond à sa position actuelle, sinon il ignore, car ça ne lui concerne pas.)

2.2.2 L'Avantage

— L'avantage pour le partage de carte, on ne partage pas notre carte en permanence alors qu'on n'est pas sûr qu'il ait un agent autour de soit, on économise ici la quantité d'information qu'un agent envoie en permanence sur le flux d'échange, grâce au système de "poke" **SayHello**. Le système d'attente maxi-

mal permet à nos agents de ne pas être indéfiniment bloqué sur une case si l'agent avec qui il a voulu communiquer est sorti du rayon de communication.

— L'avantage de cette vérification, permet de traiter facilement tous les cas possibles et rediriger vers les états correspondent aux informations traitées, et de demandé à tout moment à l'agent autour qui sont en train d'explorer ou chasser de communiquer rapidement avec un des échanges de message direct pour trouver une stratégie de blocage rapide.

— L'avantage de cette vérification rapide, est le fait de pouvoir comme son nom l'indique de vérifié rapidement si un nœud est bien un Wumpus ou non, car la vérification d'une case est vraiment très complexe et beaucoup de paramètres peut venir perturber le blocage d'un Wumpus et donc une vérification rapide et robuste, et aussi si on passe dans le mode pour trouver une stratégie de blocage du Wumpus, les coups en communication sont pas négligents, donc une vérification rapide avant est très important même s'il ne marche pas à tout les cas dû à l'interblocage possible

2.2.3 Limites, Complexité, Optimalité et Critère d'Arrêt

Les limites du partage des cartes et que malgré la vérification d'avoir un agent autour avant d'envoyer la carte, fait que parfois, un agent reçoit le message de "poke" de **SayHello**, mais entre temps ils ont bougés ou que l'un a envoyé un message, mais l'autre n'a pas envoyé et est sortie du rayon de communication donc, on ne peut pas garantir à tous les coups que l'échange se passe avec succès, les critères d'arrêt pour l'attente de **ReceiveMap** à un temps donner, si au bout d'un temps X il n'a toujours rien reçu alors il repart en exploration ce qui peut encore causer des cas où un agent envoie trop tard sa carte et l'agent qui attendait la carte est déjà parti, on perd certes en efficacité, mais on gagne en robustesse, grâce à ce point d'arrêt.

Les limites de cette vérification où l'on est quasiment sûr que ça soit un Wumpus, est comme sont noms l'indique si par erreur due à un agent qui bouge trop lentement ou qui est dans un état de sommeil ou encore l'agent bouge trop lentement par rapport aux autres agents, alors les autres agents peut considérer que cet agent est un Wumpus et accédés a cet état qui est assez lourd en termes de temps passé et d'échange d'information, en terme d'optimalité ce n'est pas le meilleur, mais en termes de robustesse on est très correcte, car même quand on est quasiment sûr d'avoir bloqué un agent on continue a essayé d'accéder sur la case où potentiellement c'est un Wumpus tout les X temps, mais on perd donc la notion de critère d'arrêt global pour tous nos agents et qui continuerait en continu tout les X temps d'essayé d'aller sur la case du probable Wumpus. Il faut toutefois noter que le comporte **StrangeWait**, peut générer un certain nombre de boucles infini en communication avec d'autres agents, dû au fait que si aucun de tous les agents on découvert auparavant le nœud où se trouve le Wumpus bloqué sur un nœud ouvert, alors à la fin de l'exploration où il restera plus que les quelques nœuds bloqué par le Wumpus qui se retrouve dessus, fait que nos agents voudront toujours y accéder et continuerons d'envoyer des demandes d'accès aux derniers nœuds ouverts.

Pour la vérification rapide, qui sur le papier est très utiles et semble ne pas avoir de point faible, car pouvant distinguer rapidement si la case qu'on essaye d'accéder soit un Wumpus ou non semble assez simple, mais la vérification est vraiment une mission très ardue, dû au bon nombre d'interactions avec les autres agents, ce qui peut créer beaucoup d'incohérence surtout lors de l'exploration où il peut rentrer en conflit avec le partage de carte, on a donc défini un certain niveau de sensibilité que les agents ont avant de lancer le protocole de vérification rapide, pour laisser le temps de faire le partage de carte, mais une fois passer en mode chasse cette sensibilité disparais et dès qu'un agent n'a pas réussi à accéder a une case il lance directement la vérification rapide, car il n'a plus le problème de conflit avec le partage de carte (qui n'est pas présent dans le mode chasse), pour qu'il vérifie plus rapidement si c'est un Wumpus ou non.

2.3 Chasse

2.3.1 Le Principe

La chasse **DumbChase** reprend les grandes lignes de l'exploration, mais avec quelque changement, tout d'abord il y a la notion de recherche du Wumpus, où il prend au hasard dans sa liste des nœuds qu'il peut accéder et il prend aussi en compte les nœuds où des agents sont stationnés pour bloquer un Wumpus pour les contournés, mais quand il ne trouve plus de chemin possible, alors il peut retourner voir l'agent qu'il l'avait bloqué auparavant pour voir si depuis il a bougé ou non, et pendant qu'il parcourt un nœud a un autre nœud qu'il a décidé d'y aller, il vérifie à chaque fois autour de lui s'il y a un bruit produit par le Wumpus, s'il perçoit du bruit, alors il est automatiquement attiré par la direction où se trouve le bruit, et si a un moment il est bloqué il déclenche la vérification rapide **Something** (pour dire Something block me), s'il pense que c'est un Wumpus alors il déclenche l'analyse du **CheckWumpusBlocked** précisé avant, où en fonction de l'État où le Wumpus se trouve il déclenche le protocole à suivre en fonction pour réussir a le coincé. Sinon tout comme l'exploration pendant la chasse, il continue ici à chasser, et en fonction des messages reçus et des protocoles reçus, il peut s'adapter pour aider un autre agent a bloqué un Wumpus, ou encore d'essayé de décaler pour laisser passer les agents qui sont en mission prioritaire pour aller bloquer un nœud. Il faut aussi préciser que si un agent perçoit un bruit au tour suivant il essayera toujours d'aller vers une case avec du bruit, mais non une case de bruit où il vient de venir, pour éviter que les agents tournent en rond. Et une information très importante déjà précisée auparavant, mais dans le mode chasse, il n'y a plus la notion de sensibilité, pour que nos agents puisse vérifier très rapidement si l'élément qui lui bloque est un Wumpus ou non.

2.3.2 L'Avantage

Tout comme l'exploration, l'avantage ici est la centralisation d'un comportement principal, pour ainsi ici chasser rapidement et en fonction des communications reçues des autres agents, il peut très rapidement prendre une décision d'ignorer si ça ne lui concerne pas ou de passer a d'autres états. Et grâce à la suppression de la

sensibilité avant de lancer une vérification de la case qui lui bloque, nos agents peuvent déterminer assez rapidement si c'est un agent ou un Wumpus en face et ainsi bloqué le plus rapidement possible Wumpus avant qu'il décide de s'enfuir.

2.3.3 Limites, Complexité, Optimalité et Critère d'Arrêt

Les limites de cette chasse, c'est qu'il repose beaucoup sur la chance de trouver un Wumpus sur le chemin, et que malgré la stratégie de blocage qui fonctionne dans la majorité des cas, ici il n'y a pas de réelles stratégies de **recherche** de Wumpus, même si en théorie tôt ou tard nos agents arriverons à trouver le Wumpus, et donc en terme d'optimalité, on n'est pas très efficace ici. Il faut aussi toutefois noter que malgré une sensibilité non présente, nos agents peuvent toujours tôt ou tard rencontrer le Wumpus, même si le Wumpus est plus rapide que nos agents, mais le blocage sera plus ardu, dû au fait que nos agents n'ont pas la notion de bloqué des nœuds cruciaux, mais ils communiquent pour directement bloquer les nœuds aux alentours du Wumpus, ce qui fait qu'il faut aussi une part de chance ici, si le Wumpus est plus rapide que nos agents. Mais il reste d'autres problèmes ou même s'il déclenche la procédure, il faut aussi compter le temps que l'agent qui va aider à bloquer le Wumpus et le temps d'arrivée sur le nœud a bloqué avant que le Wumpus décide à nouveau de bouger, ou encore que nos agents n'ont pas la notion du nombre de Wumpus sur la carte et bloque le premier qu'il voit (des golems qui ne sont pas encore bloqué), ce qui parfois peut faire qu'on est quatre agents qui bloque un même Wumpus et du coup aucun autre agent bloque le deuxième Wumpus, s'il y en a un. Mais malgré ces quelques exceptions le critère d'arrêt et d'efficacité est très correctes, car dans la plupart des cas sur des cartes de type "arbre" ou "circulaire", nos agents arrive a contourné grâce à **MoveTo** et arrive le plus rapidement possible a tous les nœuds présents aux alentours du Wumpus, et qui le bloque avec succès.

3 Conclusion

Dans l'ensemble, notre projet réalisé répond à la demande de l'objectif de base, qui est d'avoir des agents qui arrivent à se communiquer pour explorer son environnement pour ensuite bloquer le ou les Wumpus. Notre version est assez robuste et marche dans la grande majorité des cas, mais sous certaine condition, comme le fait qu'il n'est pas trop d'agent sur la carte, car on a surtout travaillé avec 2-3 agents en même temps, et au-delà ça continuerai de marcher, mais il peut y avoir des interblocages important entre les agents, ce qui peut créer une augmentation très importante des communications entre les agents avant qu'ils se débloquent, il ne prend pas en compte aussi le nombre de Wumpus sur la carte, c'est-à-dire que si les agents bloque un Wumpus à trois, ils ne vont pas chercher a bloqué les autres Wumpus, car ils seront déjà tous occupés a bloqué le Wumpus en question.
Le sujet de projet est vraiment très vaste, où à première vue, on pense que c'est un exercice simple, est en fait un exercice vraiment complexe et il y a beaucoup de paramètres qu'on peut prendre en compte. On a pensé à beaucoup de stratégie possible de blocage en prenant le maximum de paramètre en compte, mais on se heurte

rapidement à la réalité des choses, le simple fait de créer un canal de communication simple et efficace entre un grand nombre d'agents relève d'une grande complexité, on a voulu de base créer une stratégie de chasse où il y avait des protocoles d'échange entre agents avec des notions de connaissance entre les agents de savoir qui sait quoi, et ainsi pouvoir communiquer et avoir un groupe d'agents qui pourrait s'échanger et donner des rôles aux agents, par exemple sélectionner des agents pour bloquer des nœuds important, nous avons notamment lu des notions sur la théorie des graphes, tel que la notion de **dureté d'un graphe** qui permet de trouver des nœuds essentiels qui une fois bloquer, bloque toute sorti entre deux sous graphes, et utiliser cette méthode pour faire une recherche par zone et ainsi trouvé le Wumpus plus facilement tout en bloquant ces issues possible s'il était dans la zone.
Mais dans la globalité notre projet réalisé marche assez bien, et tous les comportements principaux sont présents, néanmoins dans plusieurs parties nous n'avons pas pu les améliorer et gagner en optimalité et efficacité. Ce qui est vraiment regrettable, car on a réussi à construire une base assez solide, mais qu'on n'a pas pu améliorer les différentes stratégies dans différentes situation possible, car nous avons avant tout chercher à avoir une base fonctionnelle dans la plupart des cas.

4 Lien du Guide d'utilisation

https ://dedale.gitlab.io/page/dedale/presentation/

5 Remerciement

On remercie chaleureusement toute l'équipe pédagogique qui m'ont aidé durant tout le processus de conception et de développement de mon projet durant cette période difficile en temps de pandémie mondiale et notamment mon enseignant Monsieur Cédric Herpson, pour son intérêt et son soutien, sa grande disponibilité et ses nombreux conseils durant tout le semestre !