UNIVERSITÉ PIERRE ET MARIE CURIE

FOSYMA

PROJET

Wumpus Multi-agent

SARAH LACHIHEB (GROUPE 1)

2 mai 2019

Table des matières

Table des illustrations

1 Introduction

Dans ce projet, nous devons mettre en place une coordination entre agents pour trouver et optimiser la quantité de trésor sur un environnement non connu. Ainsi, l'exploration de la carte se fera d'une manière implicite. Le jeu se termine lorsqu'il n'y a plus de trésor à collecter sur l'environnement.

Différents types d'agent sont mis à notre disposition, chacun d'entre eux possède un niveau de compétence en force et en serrurerie pour permettre l'ouverture de trésors verrouillés qui contiennent, soit de l'or, soit des diamants. Les types d'agents sont les suivants :

- Agent **Explorateur** : cet agent ne peut collecter de trésor, mais il est utile pour aider les collecteurs à déverrouiller un trésor via ses compétences.

- Agent **Collecteur** : cet agent peut collecter un seul type de trésor, en or ou en diamant. Ainsi, s'il veut collecter un trésor, il doit au préalable vérifier le type du trésor. De plus, chaque collecteur a une capacité de sac, il ne pourra pas collecter une quantité supérieure à sa capacité. Il faut préciser que chaque pick, ramassage, entraîne une perte de trésor, c'est-à-dire, plus on fait de pick, plus on perd de trésor et ce n'est pas récupérable.

- Agent **Tanker** : cet agent permet d'amasser les trésors collectés par les agents collecteurs, il a une capacité infinie. Seul les trésors transmis au tanker sont pris en compte dans le score final.

Durant la partie, un agent Golem ayant une odeur est activé sur l'environnement, son objectif est de déplacer les trésors, mais comme dit précédemment, chaque déplacement de trésor lui entraîne une perte irréversible. Il est donc important que les agents cités précédemment se coordonnent pour éviter les mouvements infligés par le Golem.

Il faut savoir pour la suite, que l'ensemble des comportements des agents sont organisés et planifiés dans un arbre de décision qui est propre à chaque type d'agent. Cet arbre de décision est contenu dans le comportement **Planification**. Chaque fois qu'un agent termine un comportement, celui-ci retourne dans la Planification et en fonction de son comportement précédemment et de son état, l'agent va être réorienté vers un comportement en cohérence avec son état.

2 Exploration

Chaque agent est introduit sur la plate-forme avec comme premier comportement : **L'exploration** qui explore son environnement avec un algorithme de parcours en profondeur avec une complexité en $\mathcal{O}(n+m)$, n le nombre de noeuds et m le nombre d'arcs.

2.1 Connaissance nécessaire

Chaque agent de la plate-forme contient trois types de carte qui sont les suivantes :

- **Carte d'exploration** : elle permet de mémoriser l'environnement, ainsi les agents partant en explorations vont mettre à jour cette carte en insérant les différents noeuds et transitions entre chaque noeud par des arcs dans un objet de type *GraphStream*. De plus, il est nécessaire de tenir à jour les noeuds ouverts, qui représentent les noeuds visités, et les noeuds fermés, qui représentent les noeuds à visiter, pour connaître la progression de l'exploration. Chaque noeud possède un label qui lui est unique et identique pour chaque agent. De plus, cette carte permet d'afficher l'évolution des noeuds découverts en rouge et visités en noir, en temps réel, pour chaque agent. Elle contient aussi différentes fonctionnalités, comme celle de calculer le chemin optimal entre deux noeuds, selon Dijsktra.

- **Carte des trésors** : elle permet de garder en mémoire l'ensemble des trésors, avec leur position, découvert au cours de l'exploration. Chaque trésor contient une date de mise à jour, un type et une quantité, un niveau de compétence nécessaire pour l'ouvrir et si le trésor est ouvert ou fermé.

- **Carte des dangers** : elle reprends le même principe que la carte des trésors, mais en contenant l'odeur du Golem et le vent.

2.2 Algorithme

Dans un premier temps, il faut savoir qu'un agent positionné à un noeud donné peut apercevoir les noeuds voisins à sa position et aussi le contenu de son noeud actuel, mais il ne peut pas voir le contenu des noeuds voisins. Ainsi, à chaque déplacement, l'agent observe son environnement et après observation du contenu de son noeud, il met à jour sa carte des dangers et sa carte des trésors. Puis, il passe à la mise à jour de sa carte d'exploration en ajoutant le noeud de sa position actuel dans les noeuds fermés et supprime son noeud de l'ensemble des noeuds ouverts. Avec l'observation des noeuds voisins, il met à jour l'ensemble des noeuds ouverts, donc à visiter. S'il a un noeud voisin à sa position qui n'est pas encore visité, il modifie son but pour y aller. Mais, s'il n'y a pas de noeud voisin en sa position, il recherche parmi les noeuds fermés le noeud le plus proche de lui et met à jour son noeud but pour y aller. L'exploration se termine lorsqu'il n'y a plus de noeuds dans l'ensemble des ouverts.

2.3 Communication

Dans la phase d'exploration, l'ensemble des agents utilisent un comportement d'EchoFlowding pour transmettre leurs cartes plus rapidement. Il faut savoir que lorsqu'un agent est en exploration, il considère les autres agents comme des explorateurs. Ainsi, il va employer le comportement d'EchoFlowding sur les autres agents tant qu'il est lui même en phase d'exploration.

2.3.1 EchoFlowding des cartes

Le comportement d'**EchoFlowding** a pour objectif de contacter les agents par rebond afin de transmettre sa carte à des agents qui n'était pas atteignable. En effet, il faut savoir que chaque agent a un rayon de communication limité donc ils ne peuvent pas transmettre un message à un agent si celui-ci est hors de porté. Ainsi l'EchoFlowding prend tout son sens, il permet plus facilement de s'affranchir de cette contrainte de distance. Le but de l'EchoFlowding est de construire un arbre pour faire remonter l'information au père et qu'ainsi le père agrège toutes les connaissances des fils et leur renvoie une carte complète, mise à jour avec toutes les informations de tous les fils.
Les agents ont deux façons d'entrer dans le comportement d'**EchoFlowding** :
- L'idée de départ était de déclencher l'EchoFlowding lorsque le nombre de noeud ouverts, donc à visiter, était supérieur à n fois le nombre de noeuds fermés. En effet, cette situation montrait que l'agent avait besoin de rapidement combler son exploration. Mais cette idée n'a pas été retenu car il ne prenait pas en considération le fait qu'un agent qui aurait beaucoup exploré n'enverrais que peu sa carte car il a beaucoup découvert, cela limite les échanges. Pareillement, si on faisait l'inverse, dans ce cas on se priverait d'informations importantes en début de découverte. Ainsi, naturellement, a été mis en place, l'EchoFlowding toute les n itérations.
- S'ils reçoivent un message de type d'**Proposition d'Echo**.

Lorsqu'ils entrent dans ce comportement, ils vérifient s'ils ont reçu une *Proposition d'Echo*. Si c'est le cas, il l'ajoute en tant que père potentiel si le *Protocole d'Echo* contenu dans le message n'est pas considéré comme ancien, et initialise son *Protocole d'Echo* avec le protocole d'Echo reçu.
En effet, lorsqu'un agent envoie une *Proposition d'Echo*, il crée un *Protocole d'Echo* qui correspond à sa racine et un numéro d'Echo correspondant au $n^{ième}$ Echo envoyé par cet agent. Un *Protocole d'Echo* est dit ancien, si l'agent qui le réceptionne a déjà reçu, du même agent, un Echo mais de numéro plus petit. Ainsi, s'il a reçu un protocole d'Echo valide, cela signifie qu'il a un père et il passe par un comportement lui imposant d'envoyer une confirmation au père afin que celui-ci

l'accepte en tant que fils. Si ce n'est pas le cas, il crée un *Protocole d'Echo* avec sa racine qui est lui même et envoie des proposition d'Echo autour de lui pour avoir des fils. Dans le cas où un agent n'a pas de père, ni de fils, il a droit à une seconde chance pour trouver un père, et si malgré cette seconde chance il ne trouve pas de père, il sort de la communication. Dans le cas où les agents ont au moins un père et/ ou au moins un fils, ils procèdent un échange de cartes.

3 Interblocage

Chaque agent a un comportement Satisfaction et de SendMap.

3.1 Comportement de Satisfaction

3.1.1 Connaissance de la satisfaction

Chaque agent de la plate-forme contient un objet de type Satisfaction qui est employé avec le comportement **Satisfaction** pour la phase d'interblocage. Cet objet contient les attributs suivant :

- **L'état de l'agent**, c'est à dire s'il est *Altruiste* ou *Egoïstes*. En effet, si l'agent reçoit un signal ayant une satisfaction plus basse que la sienne, il devient altruiste, c'est-à-dire qu'il modifie sa tâche courante, en prenant le chemin but de l'agent, et sa satisfaction pour cette tâche est égale à l'intensité du signal reçu. Mais, s'il est égoïste, il poursuit sont propre but. Un chemin but est l'ensemble des noeuds qui permettent le déplacement au but final.

- **La tâche courante**, correspondant au chemin but de l'agent. A chaque fois que l'agent décide de se déplacer, il met à jour indirectement la tache courante. Cette tâche est constituée d'un état de progression, donc s'il est immobile, en d'éloignement ou en progression, et d'une satisfaction personnelle à cette tâche. Chaque fois qu'un agent décide d'un nouveau but, la satisfaction personnelle de cette tâche est initialisée, non pas avec zéro car cela crée des cycles d'interblocage, mais entre deux bornes, 5.0 et 10.0 dans ce projet. Ainsi, un agent aura une satisfaction personnelle différente à chaque nouveau but.

- **La tâche échappatoire** permet, lorsque l'agent est altruiste, de donner un moyen d'échapper au chemin but de l'agent plus mécontent que lui. Ceci à pou but de pour résoudre l'interblocage.

- **Une base de donnée** retraçant l'ancienneté des signaux d'insatisfaction de ses voisins pour éviter de prendre en compte d'anciens signaux de répulsion.

3.1.2 Communication de la satisfaction

Le comportement de Satisfaction est employé pour débloquer n'importe quelle situation d'interblocage, que ce soit entre deux agents de même type, de différent type ou en interaction avec le Golem.

Lorsqu'un agent entre dans un comportement de **Satisfaction**, il vérifie dans sa boite aux lettres, qu'il n'a pas reçu un message de type *"Request" signal*. S'il reçoit un message de ce type, l'agent doit effectuer plusieurs vérifications, en effet il doit savoir si :

- ➢ le message n'est pas ancien.
- ➢ l'émetteur de ce message est bien voisin de lui.
- ➢ il se trouve bien sur le chemin du gêneur
- ➢ il est déjà altruiste et dans le cas où il l'est déjà il vérifie aussi si le message reçu ne vient pas de l'agent pour lequel il est devenu altruiste.
  Si toutes ces conditions sont valides, alors il prend en compte le signal reçu. S'il a reçu plusieurs signaux prioritaires(inférieur au sien) il récupère le signal minimum et le traite. Puis il

met à jour sa satisfaction personnelle en la remplaçant par celle du signal reçu. Un l'agent altruiste, respectivement égoïste, partira en comportement d'*Altruiste*, respectivement *Egoïste*.

Une fois cela effectué, on met à jour, indépendemment, la satisfaction personnelle. En effet, lorsqu'on vérifie l'état d'avancement de la tâche courante, on lui ajoute une valeur en fonction de son état de progression :

Immobile : on incrémente la satisfaction personnelle de l'agent avec une valeur comprise entre -3 et -4.

En éloignement : on incrémente la satisfaction personnelle de l'agent avec une valeur comprise entre -1 et -2.

En progression : on incrémente la satisfaction personnelle de l'agent avec une valeur comprise entre 1 et 3.

Si l'agent est *Égoïste*, et que sa satisfaction personnelle est tombée en dessous de 0, cela signifie qu'il n'a pas réussit à bouger durant un certain temps, alors il envoie un message de répulsion, sinon il part en *Planification*.

S'il est *Altruiste*, et qu'il vient de le devenir, il cherche un chemin échappatoire, s'il en trouve un, il crée une tâche échappatoire pour y aller. S'il en trouve pas, alors il continue sur le chemin but du gêneur. Dans le cas où il ne vient pas de devenir *Altruiste*, il y a deux cas :

- ➢ Dans le premier, il a réussi à bouger en étant en recherche échappatoire, cela veut dire qu'il n'est plus en situation d'interblocage, il passe donc en mode *Égoïste* et recherche un nouveau but en partant en *Planification*.
- ➢ Dans le second, il a réussi à bouger en étant sur le chemin but du gêneur, alors il recherche de nouveau un chemin échappatoire à partir de son noeud courant.

Remarque : l'agent n'a pas le droit de retourner sur ces traces, cela empêche les cycles. On peut le voir comme une liste tabou, on lui refuse de repasser par un chemin qu'il a déjà emprunté.

Par contre si l'agent est un ancien *Altruiste* et qu'il n'a pas pu bouger, on a deux cas :

- ➢ Soit il était en recherche échappatoire, alors signifie qu'il y avait un agent sur le noeud échappatoire qu'il convoitait, et dans ce cas il regarde s'il peut avoir un autre noeud échappatoire. Dans le cas où il n'en trouverait pas, alors il continue sur le chemin but du voisin.
- ➢ Soit il doit poursuivre sur le chemin but de l'agent pour lequel il est devenu *Altruiste* et à cause du blocage il propage le signal de répulsion.

Remarque : Une fois arrivé à la fin du chemin but du gêneur, sans avoir pu trouver d'échappatoire, alors l'agent passe automatiquement en mode *Égoïste* et recherche un nouveau but.

Le principe de la satisfaction est d'introduire une priorité sur les autres agents en fonction de leur niveau de mécontentement. Si un agent n'arrive pas à bouger mais qu'il a une satisfaction positive, celui-ci va patienter mais dès lors qu'il passe en négatif, il va envoyer un message de répulsion. Si deux agents sont en négatif, c'est celui qui a la satisfaction la plus négative qui prime et l'autre devient altruiste, celui-ci doit trouver un nouveau chemin sans déranger son voisin qu'il gêne. Mais si un agent atteint un seuil de mécontentement alors il abandonne son but et cherche un chemin ne contenant pas le noeud sur lequel il n'a pas pu accéder.

3.2 Send Map (Exploration)

Le comportement **SendMap** est utilisé lorsqu'un agent ne peut pas se déplacer sur le noeud voulu. En effet, lorsque l'agent ne peut pas bouger, il envoi un message du type *"Inform"* avec le protocole *SEND_MAP* ayant pour contenu ses trois cartes et l'envoi à l'ensemble des agents inscrit dans le DFS. Puis, l'agent attend durant 500 ms la réception d'un message de type *"Inform"* avec le protocole *SEND_MAP*. Dans le cas où il reçoit un message de ce type, il met à jour ses cartes. En fonction de son identifiant, celui qui est prioritaire prend le noeud ouvert le plus proche à explorer, et le second prend le noeud ouvert le plus loin, de leur position respective. Cette situation permet de régler les interblocages entre explorateurs tout en partageant les cartes. Mais en attendant la réponse de l'agent, pour permettre de résoudre l'interblocage sans utiliser la satisfaction, il y a plusieurs exemplaires de messages de protocole *"SEND_MAP"* qui sont envoyés. Il aurait été intéressant de mettre en place une procédure de mémorisation d'envoi de message en cas d'interblocage pour limiter le nombre de messages envoyé.

4 Collecte des trésors

4.1 Comportement de recherche position Silo pour le Tanker

Ce comportement est utilisé par les agents Tanker. Lorsque l'agent Tanker a terminé son exploration, il trouve une position silo correspondant à son identifiant, ainsi si c'est le premier Tanker créé sur la plate-forme Jade, il récupère le premier noeud de plus haut degré dans l'environnement ayant le label le plus petit en terme de chaîne de caractère ASCII. Le but d'avoir un emplacement unique pour chaque Tanker. La position Silo trouvée, le tanker ne bouge plus de cette position et modifie le niveau de sa satisfaction personnelle de la tâche courante à zéro. En effet cela permet au Tanker de facilement se déplacer s'il gêne. Lorsqu'il se trouve à la position Tanker, il fait un Broadcast de sa position Silo à tous les agents, avec pour protocole *"Position Silo"* de type *"Inform"*. Il vérifie aussi, s'il reçoit un message de type *"Confirm"* de protocole *"Position Silo"*. Si c'est le cas l'agent Silo sait qu'il doit plus lui envoyer sa position car l'agent a pris connaissance de cette dernière. De plus, l'agent Tanker vérifie s'il a reçu un message de type *"Inform"* de protocole *"Carte environnement"*. Ce message lui permet de récupèrer les cartes fournit par les agents Explorateurs et Collecteurs, dans le but que le Tanker soit utilisé comme borne d'information. En effet lorsqu'il reçoit un message de ce type, il répond à l'expéditeur de ce message avec sa carte, pour tenir à jour les informations des autres agents de la carte.

4.2 Comportement de recherche de recherche de Silo pour tout les agents sauf le Tanker

Ce comportement est employé par l'agent Explorateur et l'agent Collecteur. Lorsque les agents ont besoin de connaître la position des silos, ils entrent dans ce comportement. Les agents recherchent dans les pages jaunes du DFS tous les agents Tanker. A l'aide de l'algorithme permettant de trouver les emplacements potentiels des Tankers, ils font le tour de ces positions pour aller s'identifier auprès de ces derniers et vérifier s'il se trouvent bien à leurs positions.

4.3 Comportement de recherche de trésor individuel

Ce comportement est utilisé par les agents de type Collecteur. Il a pour objectif de déterminer les actions que doivent entreprendre le Collecteur, de ce manière optimiser. Lorsque l'agent termine son exploration, il entre dans le comportement de **PDMBehaviour**. Si l'agent n'a pas d'action en cours, il effectue une résolution d'un processus de décision Markovien qui lui donne l'action à exécuter. Les différentes actions peuvent le mener à plusieurs comportement qui sont les suivants :

Action Interblocage : Cette action est commune à l'agent Explorateur et Collecteur. Si l'agent est en gestion d'interblocage en étant devenu altruiste, alors il ne va pas exécuter le PDM et continue de gérer son interblocage.

Action Tanker : Le PDM peut diriger l'agent vers un **Tanker** si son sac est trop plein pour collecter n'importe quel trésor. Si tous les trésors sont verrouillés et qu'ils demandent une compétence supérieure à celle de l'agent alors il entre dans le comportement de **recherche de trésor collectif**.

Action collecte trésor individuel : cette action est utilisable que par l'agent collecteur. Si dans sa carte des trésors, il n'y a que des trésors de son type qui sont ouverts ou ouvrables par l'agent avec ses compétences, alors le PDM l'oriente vers le trésor le plus propices à lui apporter le plus de richesse en peu de pick, avec une distance raisonnable et avec un probabilité de présence forte.

4.3.1 Algorithme pour le Processus de décision Markovien

Un processus de décision markovien est un quadruplet <S, A, T, R> qui se définit par :
S : Le nombre fini d'états. Dans le cadre de ce projet, les états sont :

- ➢ La position qui correspond à la position de l'agent lors de la création du PDM.
- ➢ La position de chacun des trésors.
- ➢ la position de chacun des Tankers.
- ➢ Un état vide, aussi appelé état absorbant.

A : Un ensemble d'actions. Ici une action est d'aller à l'un des trésors définit dans les états ou pareillement avec les Tanker.

$T : S \times S \times A \rightarrow$ L(S) : La fonction de transition est modélisée par une matrice à trois dimensions. La matrice de transition calcule pour chaque $état_i$ la probabilité d'aller à un autre $état_j$ en exécutant une certaine action. Les probabilités de la matrice sont déterminées de la manière suivante :

- ➢ La position qui correspond à la position de l'agent lors de la création du PDM.
- ➢ On récupère l'ensemble des trésors présent sur la carte.
- ➢ Dans cette ensemble, on supprime les trésors ne pouvant pas être collectés par l'agent collecteur en question, soit car il n'a pas les compétences requises, soit car il n'est pas du bon type, soit car il n'a plus de place dans son sac. On obtient à la fin un ensemble de trésors valides.
- ➢ On trie les trésors par date de mise à jour, du plus récent au plus ancien, car on souhaite donner une plus grande importance à l'action d'aller à un trésor plus récent.
- ➢ On met à jour la matrice de probabilité en fonction de chaque trésor par le calcul suivant :

$$T[EtatPosition][tresor_i][actionTresor_i] = \frac{nbTresorValide}{\sum_{i=1}^{n} i}$$

Avec n le nombre de trésors valides. Cela nous permet d'obtenir une probabilité normalisée, pour une action, d'aller sur chacun des trésor. Plus le trésor est récent plus il aura une plus grande probabilité.

- ➢ La matrice de transition est initialisée avec 0 et seules les cases où les transitions sont possibles, selon l'action, sont changées à l'aide de la formule suivante, pour les trésors :

$$T[EtatPosition][EtatVide][actionTresor_i] = 1 - T[EtatPosition][tresor_i][actionTresor_i]$$

Cependant la formule est modifiée pour les tankers car il n'y a pas de risque de tomber sur un état vide :

$$T[EtatPosition][EtatTanker_i][actionTanker_i] = 1$$

$R : S$ x $A \rightarrow$ R : La fonction de récompense pour chaque état est définie de la manière suivante :
- L'état Position a une récompense de 0.
- L'état Vide a une récompense de 0.
- La récompense attribuée pour chacun des trésors de l'environnement est déterminée par différentes étapes :
1) Si le collecteur est en mesure de collecter ce type de trésor et qu'il possède encore de l'espace disponible dans son sac, il détermine le nombre de Pick nécessaire pour collecter tout le trésor. A partir de là, il incrémente la récompense de cet état par le calcule suivant :

$$recompense+ = \frac{(\frac{Qressource}{\sum_{i=1}^{n} Qressource_i} * bonus.TresorNormalise)}{nbPcik}$$

avec $nbPick$ le nombre de pick, $Qressource$ la quantité de trésor, $nbTresor$ le nombre de trésor sur la carte.
2) Si le trésor est fermé, on attribut une récompense en fonction de la capacité de l'agent. Si l'agent a la force pour ouvrir le trésor, on lui incrément la récompense de :

$$recompense+ = bonusForce * tresor.getStrenght()$$

Si l'agent a la compétence de serrurier pour ouvrir le trésor, on lui incrément la récompense de :

$$recompense+ = bonusSerrurier * tresor.getLockpicking()$$

Par contre, si le trésor est déverrouillé, on additionne à la récompense le calcul suivant :

$$recompense+ = bonusForce * bonusSerrurier$$

3) On prend en compte la distance au trésor, plus le trésor est près plus la récompense sera grande. Pour ce faire, on détermine au préalable la plus longue distance entre tous les trésors et on fait ce calcul :

$$recompense+ = distMaxAllTresor - tresor.nbNoeudChemin()$$

- La récompense attribuée à chacun des Tankers dépend de la distance de l'agent à chacun des Tankers. Ainsi, il est nécessaire de déterminer chaque plus court chemin pour aller au Tanker avec un Dijkstra puis on mémorise la plus longue distance entre tous les Tankers. Cela permet de donner un score plus grand pour le Tanker le plus près de l'agent via ce calcul :

$$recompense+ = (distMaxAllTanker - tanker.nbNoeudChemin()) + 1$$

4.3.2 Programmation linéaire résolu par CPLEX_Studio128

Afin de résoudre le modèle de processus de décision Markovien posé précédemment, on utilise le programme linéaire suivant :

Algorithm 1 Formulation Programme linéaire

Input ↓ :
S ▷ les états
R ▷ la matrice de récompenses
T ▷ la matrice de transitions
Output ↑ :
d(S) ▷ décision pour chaque état

1: $min_V \sum_{s \in S} V(s)$
2: $s.c.$
3: $V(s) \leq R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s') \qquad \forall s \in S, \forall a \in A$
4: **for** $s \in S$ **do**
5: $\quad d(s) \rightarrow choix[argmax_{a \in A} \quad \{R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')\}]$
6: **end for**
7: **return** $d(s)$

Pour résoudre le PDM ainsi formé on utilise les équations de Bellman qui sont un moyen assez simple de trouver une politique stationnaire, c'est à dire qui attribue une seule et unique politique à chaque état (donc pas en fonction du temps). Ces équations peuvent se résoudre via deux algorithmes connu, policy et value iteration. Sans surprise value iteration a été le plus long à converger que policy. C'est donc ce dernier qui a été implémenté ici. Aussi, la formation par PL est plus rapide qu'une approche classique.

4.4 Comportement de recherche de trésor collectif

Ce comportement est utilisé par les agents de type Collecteur, lorsqu'ils n'ont plus de trésor individuel à collecter mais qu'il reste des trésors à déverrouiller, et par les agents Explorateurs, lorsqu'il reste des trésors à déverrouiller.

Un agent Explorateur se dirige en priorité vers un trésor déverrouillé qui demande le moins de compétence, si égalité alors avec le moins de ressources. Un agent Collecteur va, dans un premier temps, chercher les trésors qu'il peut directement collecter. Une fois cela effectué, il part en recherche de trésors collectif, de la même manière que l'explorateur. Dès qu'un coffre a été déverrouiller, correspondant à son type, il part en recherche de trésor individuel et entre dans le comportement de **PDMBehaviour**.

Lorsque les agents sont dans le comportement **Recherche de trésor collectif**, s'ils n'arrivent pas à bouger, ils envoient un message de type *"Propose"* de *Protocole "Help"* pour proposer leur aide. Ils patientent durant $500ms$ et vérifient dans leur boite aux lettres s'ils ont une acceptation(à leur proposition d'aide) de type *"ACCEPT_PROPOSAL"* de **protocole "HelP"**. S'ils reçoivent une acceptation, ils passent en attente de trésor, c'est-à-dire qu'il modifient leur satisfaction personnelle en la fixant à zéro. Étant donné que la tâche courante de l'agent était d'aller au trésor collectif, s'il décide d'attendre à cette position toute en gardant la tâche courante active alors l'immobilité de l'agent sera pénalisé par une baisse de sa satisfaction personnelle. Si elle tombe en dessous du seuil critique, alors il abandonnera sa tâche, ce que l'on ne souhaite pas. De plus on ne souhaite pas qu'il envoie des messages de répulsion, donc on ne veut pas que sa satisfaction personnelle descende en dessous de zéros. Aussi cela permet qu'ils soient facilement déplaçable en cas d'interblocage.
Maintenant l'agent passe en attente de réception d'ouverture de coffre.
Lorsque le coffre a été ouvert, car il y a eu assez d'agents pour l'ouvrir, il propage le message d'ouverture aux agents auprès de lui. Si le coffre qui a été ouvert correspondait à son coffre cible alors met à jour sa carte. De même, si l'agent reçoit une proposition d'aide, il vérifie si l'agent en question

a bien pour objectif le même coffre que lui et si il est voisin de sa position. Si tel est le cas alors il lui envoi un message d'acceptation d'aide. L'agent qui se trouve sur le trésor essaye à chaque itération de l'ouvrir jusqu'à qu'il y ait assez de compétence. Lorsqu'il arrive à ouvrir le coffre, il envoi aux agents à proximité la validation d'ouverture de trésor avec sa carte actuel pour un meilleur partage d'information.

4.5 Comportement d'exploration de trésor perdu

Ce comportement est employé par les agents Explorateur et Collecteurs. Dès lors que l'agent n'a plus de trésor à ouvrir ou à déverrouiller, il part en comportement d'**exploration de trésor perdu**. Ce comportement consiste à rechercher d'éventuels trésors déplacés par le Golem en débutant une nouvelle exploration de l'environnement. Toutes les n itérations, l'agent fait le tour des tankers pour voir s'il n'y pas de trésor qui a été découvert entre temps. Si un agent en exploration de trésor perdu découvre un trésor, alors le traitement dépend du type de l'agent. Si c'est un agent collecteur, il regarde s'il peut le collecter directement, si ce n'est pas le cas il retourne informer l'ensemble des tankers et part de nouveau en recherche de trésor collectif. Si l'agent est un Explorateur, il part directement informer l'ensemble des Tanker et entre dans le comportement de recherche trésor collectif.

5 Conclusion

Il aurait été intéressant, comme suite de ce projet, de prendre le Tanker comme une borne d'information. Lorsqu'un agent ne peut ouvrir seule un trésor il pourrait être re-dirigé vers un agent Tanker afin de lui demander quel trésor il peut aller collecter et avec qui. Le Tanker, en fonction des agents qui lui demande de l'aide, aurait agrégé l'ensemble des compétences de ces agents et grâce au PDM il formerait des groupes à envoyer sur un trésor particulier de manière optimisée. Aussi, les coalitions entre agents aurait pu être formé par le Tanker à l'aide d'un algorithme dérivé de la valeur de Shapley, cela dans le but d'avoir un nombre d'agent optimal sur un trésor.
On voit rapidement les problèmes majeurs que peuvent entraîner une telle stratégie. Comment faire si les agents choisissent des tankers différent et qu'il n'y a pas assez d'agents autour du Tanker pour aller ouvrir un trésor ? Comment gérer l'affluence des agents autour des tankers sans bloquer les agents qui veulent déposer leur trésor ?

Références

[1] Olivier Simonin-Jacques Ferber. Modélisation des satisfactions personnelle et interactive d'agents situés coopératifs.

Annexes

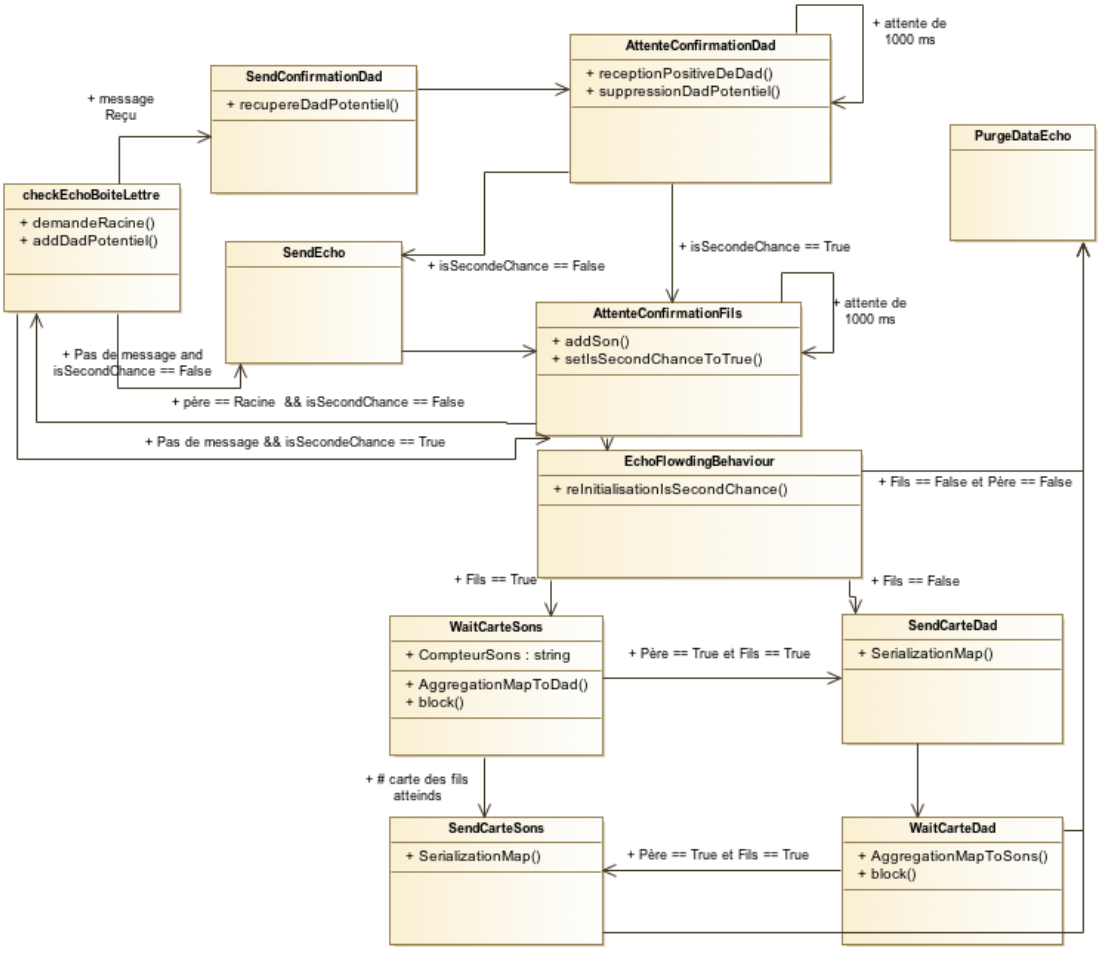

FIGURE 1 – Diagramme UML EchoFlowding

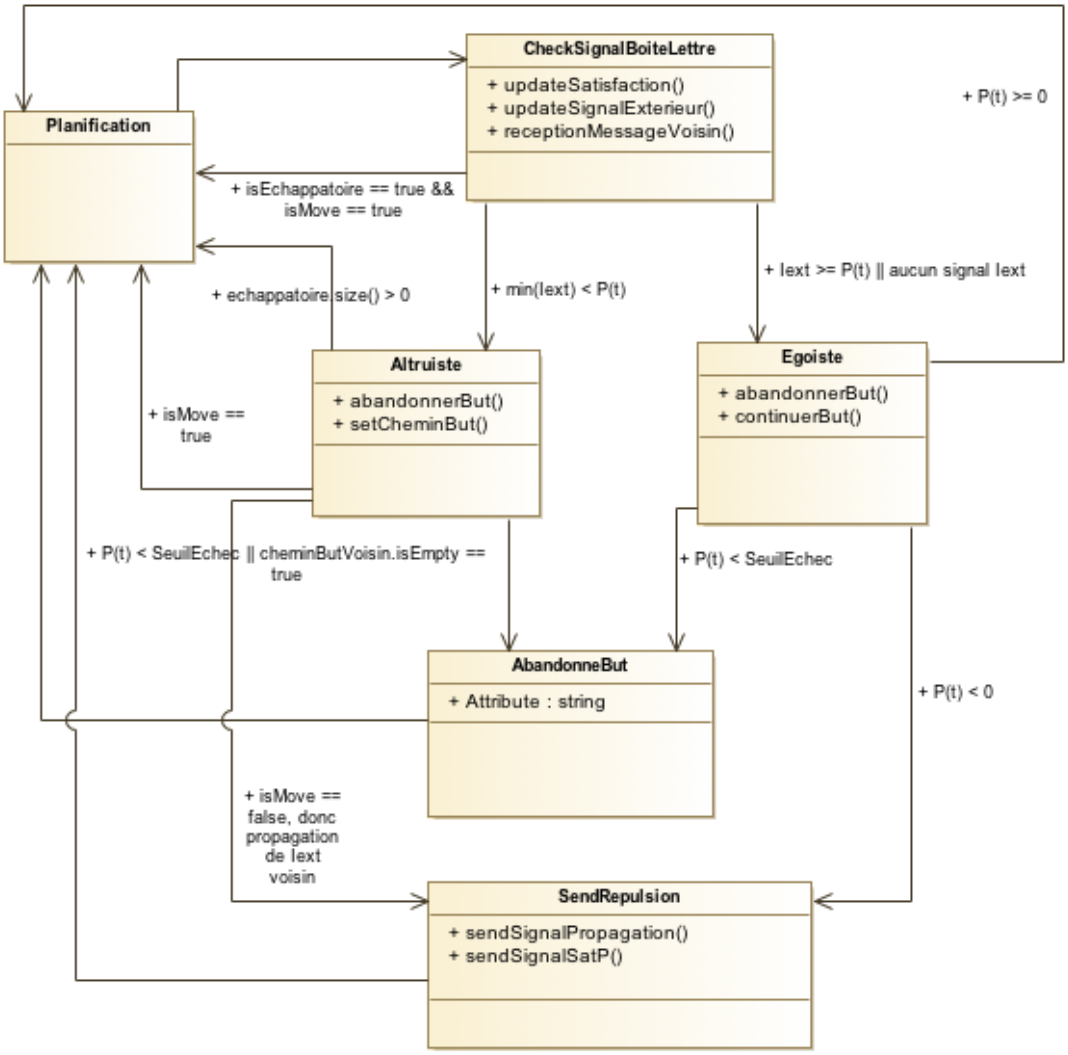

FIGURE 2 – Diagramme UML Satisfaction

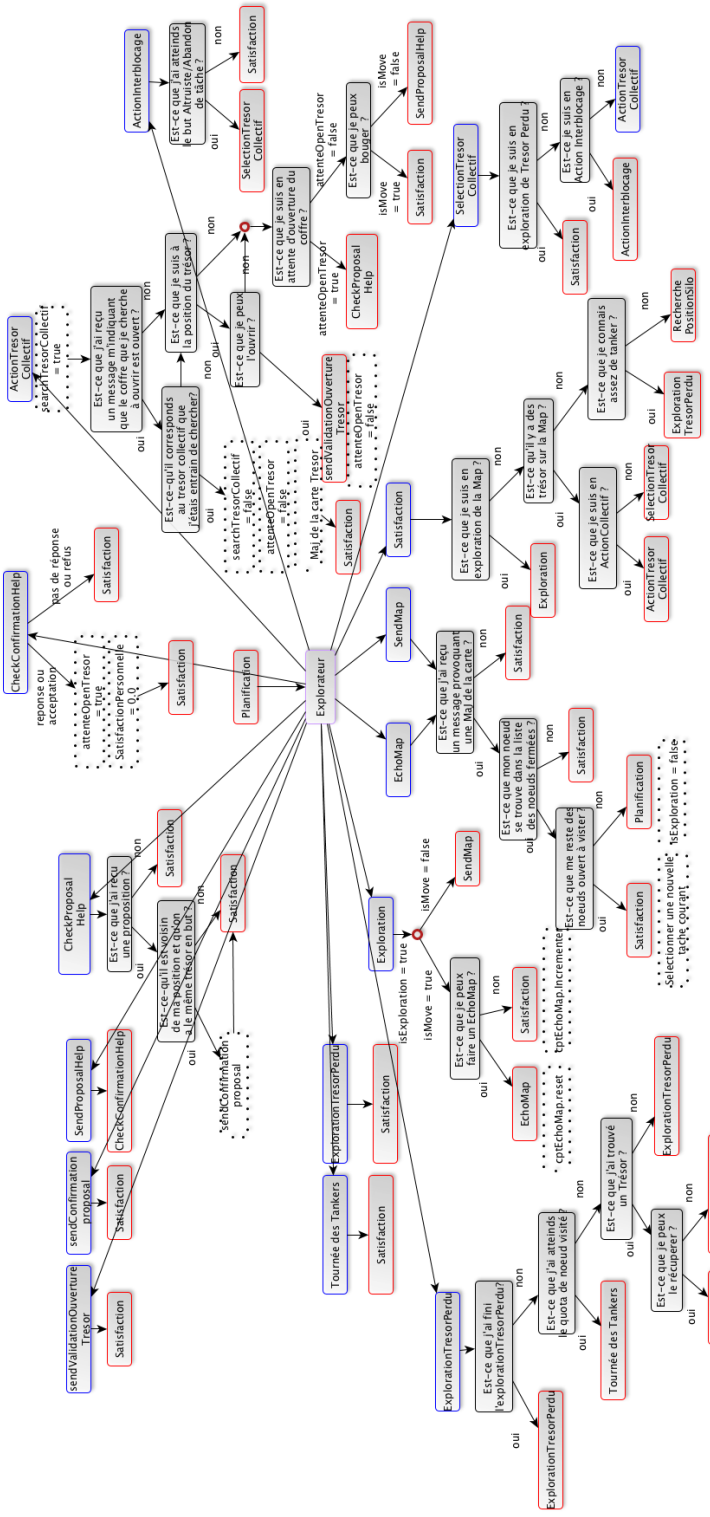

FIGURE 3 – Architecture de l'agent Explorateur

FIGURE 4 – Architecture de l'agent Collecteur

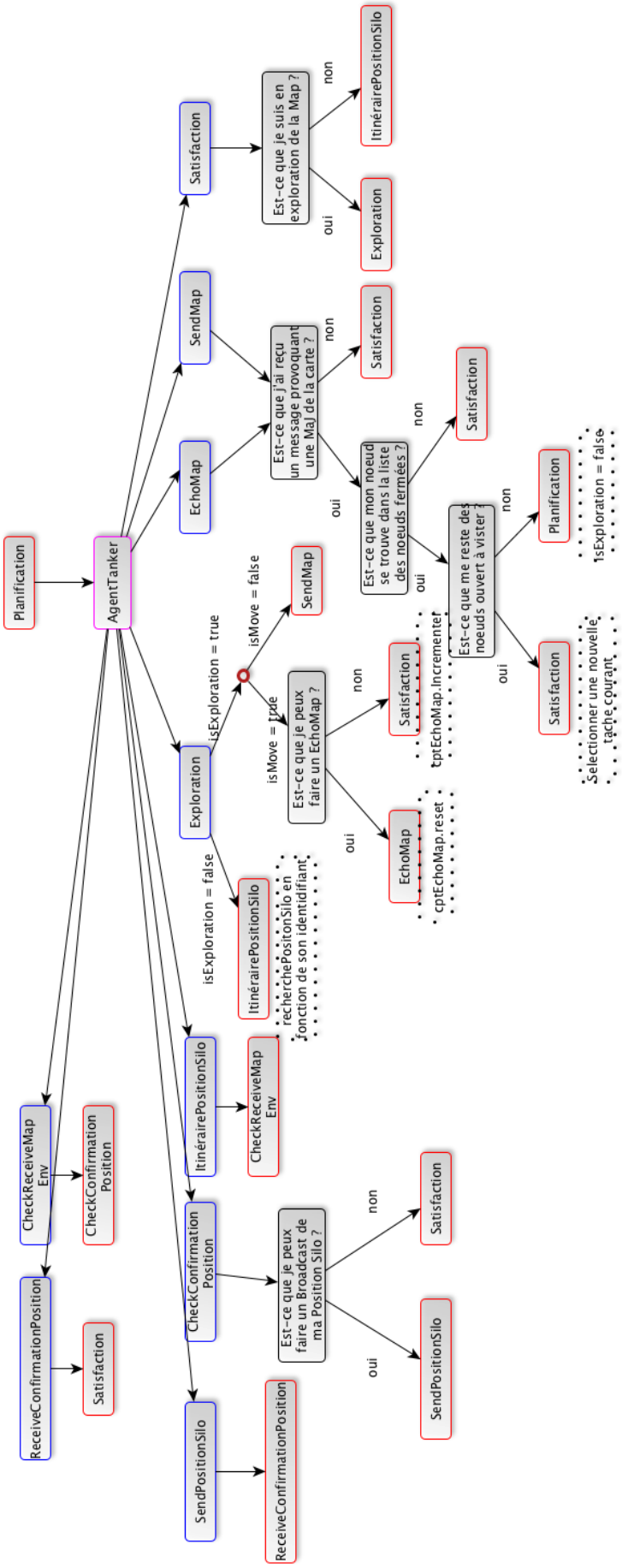

FIGURE 5 – Architecture de l'agent Tanker