Projet FoSyMa - Dedale

Michelle Song, Camélia Bouali

8 mai 2025

Table des matières

1 Introduction

Le projet Dedale, réalisé dans le cadre de l'UE de FoSyMa (Fondements des Systèmes Multi-Agents), porte sur la conception et l'implémentation d'un système multi-agent coopératif inspiré du jeu *Hunt the Wumpus*. L'objectif principal est de créer un ensemble d'agents autonomes capables d'interagir efficacement dans un environnement inconnu et dynamique, en maximisant la collecte de ressources (trésors), tout en faisant face à des défis liés à l'exploration, la coordination, la communication et la gestion de contraintes locales.

Ce projet met en jeu plusieurs problématiques classiques des systèmes multi-agents, telles que la gestion de l'information locale, le partage de connaissances entre agents ou encore la prise de décision distribuée. La plateforme Dedale, basée sur JADE, fournit un cadre de simulation où les agents évoluent dans un environnement représenté sous forme de graphe, avec une visibilité limitée, des actions asynchrones et des rôles différenciés.

Le système à concevoir vise à permettre aux agents de coopérer intelligemment pour cartographier leur environnement, identifier et collecter des ressources, et optimiser leurs déplacements et leurs interactions, tout en prenant en compte la présence d'un adversaire mobile : le Golem. Ce travail implique une série de choix algorithmiques et architecturaux tenant compte des ressources limitées des agents, de leur champ de vision restreint et des contraintes de communication locale.

Ce rapport introduit l'architecture de notre solution, les comportements développés pour les agents, ainsi que les stratégies mises en place pour répondre aux différents défis du projet. Il présente les choix techniques et méthodologiques retenus, les algorithmes employés, et propose une analyse de leurs performances, de leurs limites, ainsi que des pistes d'amélioration.

2 Architecture générale du système

2.1 Vue d'ensemble de l'architecture

2.1.1 Structure des dossiers et des fichiers

Le code source du projet est organisé de manière modulaire, ce qui reflète les différents comportements et rôles des agents dans le système multi-agent. La structure suivante a été mise en place pour faciliter la gestion et l'extension du projet :

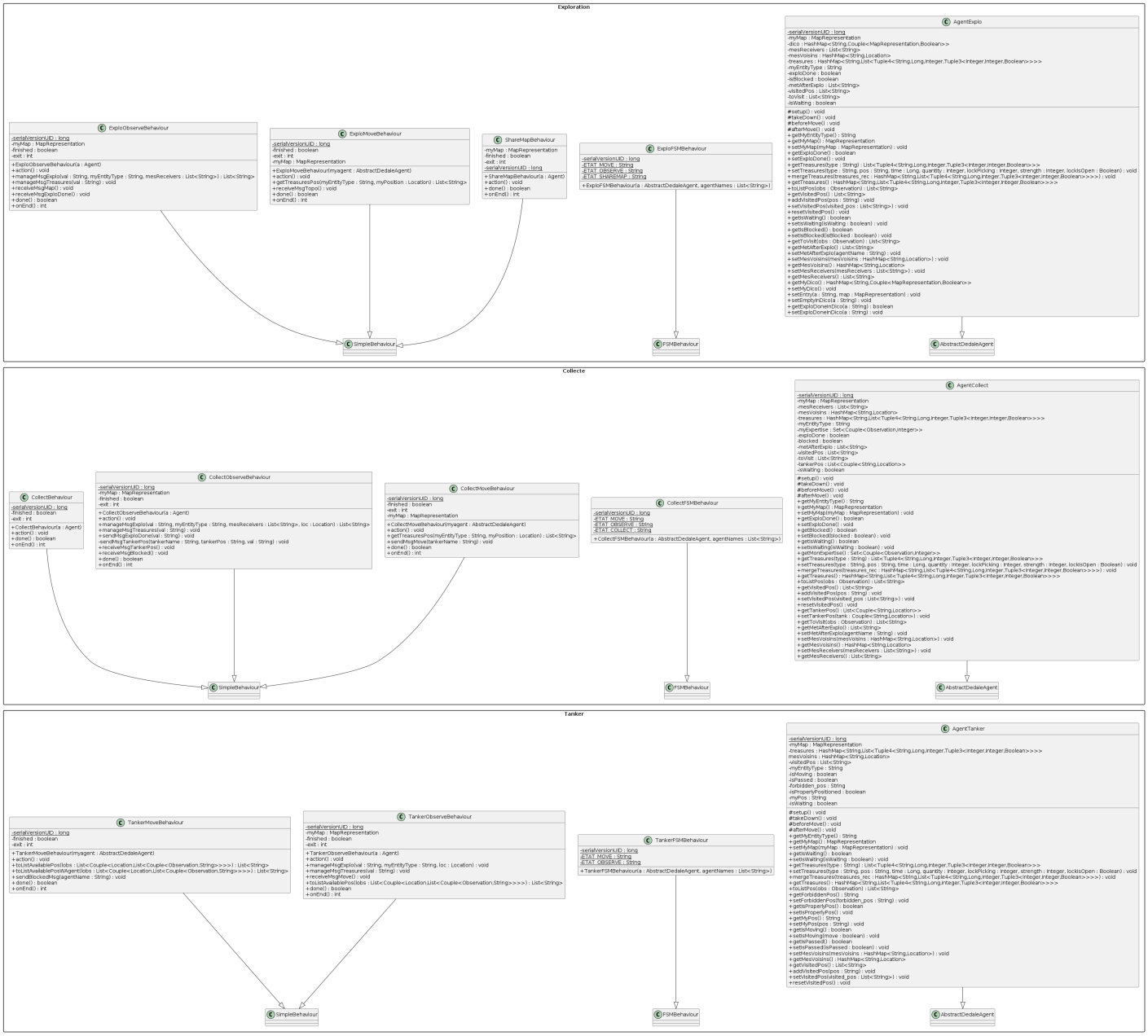
- **Dossier** `mesAgents` : Ce dossier contient les classes principales représentant les agents dans le système, chacune correspondant à un type d'agent spécifique (Collecteur, Explorateur et Stockeur) :
  - `AgentCollect.java` : Classe représentant l'Agent Collecteur, responsable de la collecte des trésors.
  - `AgentExplo.java` : Classe représentant l'Agent Explorateur, dédié à l'exploration de l'environnement.
  - `AgentTanker.java` : Classe représentant l'Agent Stockeur, chargé de gérer le stockage.
- **Dossier** `mesBehaviours` : Ce dossier contient les comportements spécifiques des agents, subdivisé en trois sous-dossiers correspondant à chaque type d'agent. Chaque sous-dossier comprend des fichiers Java pour définir les comportements des agents.
  - **Sous-dossier** `CollectBehaviours` : Comporte les comportements relatifs à l'Agent Collecteur.
    - `CollectBehaviour.java` : Classe abstraite qui définit le comportement général de collecte.
    - `CollectFSMBehaviour.java` : Comportement implémentant la machine à états finis (FSM) de l'Agent Collecteur.
    - `CollectMoveBehaviour.java` : Comportement de déplacement de l'Agent Collecteur.
    - `CollectObserveBehaviour.java` : Comportement d'observation de l'environnement pour détecter des trésors.
  - **Sous-dossier** `ExploBehaviours` : Contient les comportements relatifs à l'Agent Explorateur.
    - `ExploFSMBehaviour.java` : Comportement implémentant la FSM de l'Agent Explorateur.
    - `ExploMoveBehaviour.java` : Comportement de déplacement de l'Agent Explorateur.
    - `ExploObserveBehaviour.java` : Comportement d'observation de l'environnement pour l'Agent Explorateur.
    - `ShareMapBehaviour.java` : Comportement pour partager la carte locale de l'Agent Explorateur avec les autres agents.
  - **Sous-dossier** `TankerBehaviours` : Contient les comportements relatifs à l'Agent Stockeur.
    - `TankerFSMBehaviour.java` : Comportement implémentant la FSM de l'Agent Stockeur.

— `TankerMoveBehaviour.java` : Comportement de déplacement de l'Agent Stockeur.
— `TankerObserveBehaviour.java` : Comportement d'observation de l'environnement pour l'Agent Stockeur.

2.1.2 Explication de l'architecture

Cette organisation modulaire permet de séparer clairement les rôles et comportements des agents. Chaque type d'agent a ses propres classes qui gèrent ses actions principales. Les comportements sont intégrés dans les classes des agents, et les machines à états finis (FSM) sont implémentées pour chaque agent, garantissant une gestion prévisible et réactive de leurs actions en fonction des événements observés dans l'environnement.

2.1.3 Diagrammes UML



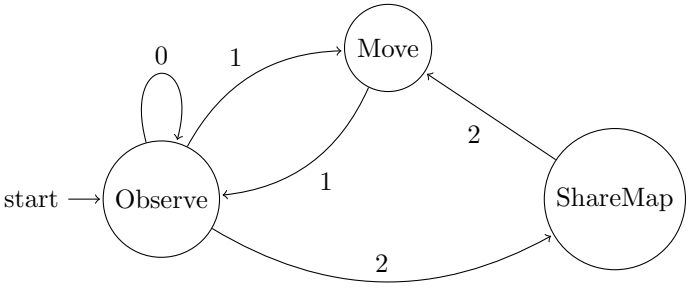
2.2 Machine à états finis (FSM)

Pour modéliser les comportements des agents dans notre système, nous avons décidé d'implémenter des FSM. Chaque type d'agent possède un ensemble d'états et de transitions qui définissent son comportement en fonction des événements qui se produisent dans l'environnement.

2.2.1 Agent Explorateur

L'agent Explorateur, chargé de l'exploration, suit un comportement en trois états :

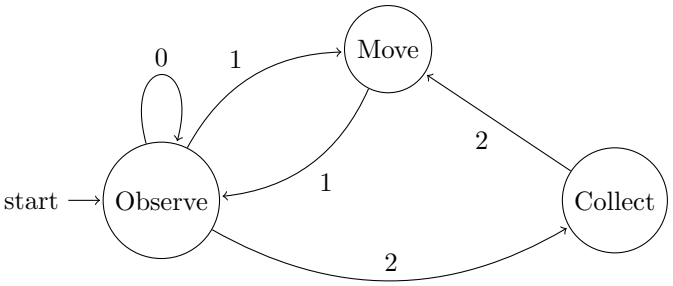
- **Observe** : L'agent observe son environnement. Il reste dans cet état s'il est bloqué, c'est-à-dire si des agents occupent les noeuds voisins. Il peut passer à **ShareMap** pour partager sa carte locale avec les autres agents détectés.
- **Move** : L'agent se déplace pour explorer l'environnement, sauf s'il est bloqué.
- **ShareMap** : Si des agents voisins sont détectés, l'agent partage sa carte locale avec eux.



2.2.2 Agent Collecteur

L'agent Collecteur, dédié à la collecte des trésors, adopte également une FSM en trois états :

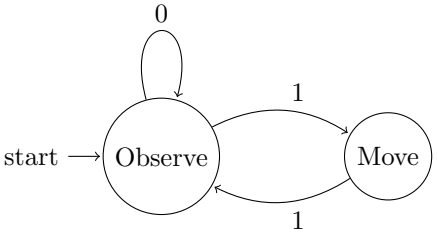
- **Observe** : L'agent observe son environnement pour détecter des trésors. S'il en trouve un et peut le ramasser, il passe à **Collect**. Sinon, il passe à **Move** ou reste dans **Observe** s'il est bloqué par d'autres agents.
- **Move** : L'agent se déplace afin de trouver des trésors.
- **Collect** :L'agent ramasse le trésor lorsqu'il atteint un nœud contenant un trésor, à condition qu'il ait les capacités nécessaires pour le faire.



2.2.3 Agent Tanker (Stockeur)

L'agent Tanker dispose de deux états :

- **Observe** : L'agent reste dans cet état et continue d'observer son environnement. Il passe à Move si sa position bloque d'autres agents, afin de libérer le passage.
- **Move** :L'agent se déplace vers une position où il ne gênera pas les autres agents. Une fois déplacé, il repasse à **Observe**.



3 Choix des stratégies

Cette section présente les stratégies mises en place pour l'exploration, la communication, la coordination et la collecte des trésors par les agents.

3.1 Exploration de l'environnement

Initialement, tous les agents Explo et Collect patrouillent l'environnement.

Les agents Explo ont un dictionnaire avec pour clé le nom des agents Collect qu'ils ont rencontrés, et pour valeur, un couple de MapRepresentation et d'un booléen. Ils correspondent respectivement à la carte contenant les nouveaux nœuds découverts et l'attribut exploDone, qui est mis à vrai si l'agent (qui porte le nom correspondant à la clé) a fini d'explorer ou faux sinon :

— Rencontre entre un agent Explo et un agent Explo : ils s'envoient tous les deux leurs dictionnaires, et les fusionnent.

— Rencontre entre un agent Explo et un agent Collect : l'agent Explo envoie la valeur associé à la clé de l'agent dans son dictionnaire. L'agent Collect, reçoit ce bout de carte et la fusionne avec la sienne.

Les agents Collect vont patrouiller l'environnement tant que leur liste de trésors est vide. Cette liste de trésors est un de leurs attributs privés, et correspond à la liste de tous les trésors trouvés. Les agents Explo aussi détiennent cette information. Ainsi, pour les agents Collect, la liste contient les trésors associés à leur type (par exemple, Gold) et pour les agents Explo, il y a tout type (Gold ET Diamond).

Pour explorer l'environnement, ils commencent par les nœuds ouverts les plus proches. S'il ne reste plus de nœuds ouverts, alors ils parcourent la liste des trésors.

- **Forces** :
  - **Adaptabilité** : Les agents peuvent s'adapter à leur environnement de manière autonome, ce qui leur permet d'explorer de manière flexible.
  - **Efficacité spatiale** : Grâce à la patrouille de plusieurs agents, une couverture rapide et complète de l'environnement est possible, optimisant ainsi le temps d'exploration global.
  - **Partage d'informations** : La fusion des cartes entre les agents permet de partager les informations sur les zones explorées, réduisant ainsi la redondance dans l'exploration.
- **Limites** :
  - **Redondance dans l'exploration** : Si les agents Explo ne se coordonnent pas efficacement, certains peuvent explorer des zones déjà couvertes, ce qui entraîne une perte d'efficacité.
  - **Dépendance aux interactions** : La performance de l'exploration dépend fortement de la rencontre et de la fusion des cartes entre agents. L'absence de rencontres entre agents pourrait ralentir la couverture de l'environnement.
  - **Communication limitée** : Bien que les agents partagent des informations lors de leurs rencontres, cela n'est possible que lorsque deux agents se croisent. Les informations ne sont pas propagées instantanément à l'ensemble du groupe d'agents.
- **Complexité** :
  - **Temps** : La complexité temporelle dépend de la taille de l'environnement et du nombre d'agents. Plus l'environnement est vaste et plus le nombre d'agents est élevé, plus l'exploration prendra de temps.
  - **Mémoire** : Chaque agent doit stocker les informations de la carte qu'il découvre ainsi que celles des autres agents qu'il rencontre. La mémoire nécessaire augmente donc avec la taille de l'environnement et le nombre d'agents.
  - **Critère d'arrêt** : Dans notre implémentation, l'exploration ne s'arrête jamais totalement. Les agents continuent à explorer l'environnement tout au long de l'exécution du système afin de :
    - Réagir à de nouveaux événements (ex. : déplacement des trésors par le Golem).
    - Offrir une assistance potentielle à d'autres agents encore en phase d'exploration ou de collecte.

3.2 Collecte des trésors

Ce sont les agents Collect qui s'occupent de la collecte des trésors. Lorsque leur liste de trésors contient au moins un élément, ils vont vers cette position. À chaque rencontre avec un autre agent, ils partagent leur liste pour pouvoir la mettre à jour et garder l'information la plus récente.

Ils regardent seulement les trésors non vides (quantité différente de 0). Une fois que leur capacité maximale est atteinte, ils vont vers le tanker, s'ils connaissent sa position. En effet, ils partagent aussi la position du tanker si elle est connue.

Par ailleurs, nous avons décidé d'attribuer une position fixe au tanker pour pouvoir rendre efficace le vidage des sacs. Initialement, s'il est placé dans un "couloir" (2 noeuds atteignables) ou un "trou" (1 noeud atteignable) alors il se déplace jusqu'à se retrouver dans un carrefour (au moins 3 noeuds atteignables). Cela est pour simplifier le décalage du tanker s'il bloque le passage d'un autre agent.

- **Forces** :
  - **Mise à jour dynamique** : Les agents Collect partagent leurs listes de trésors, ce qui permet une actualisation régulière des informations.
  - **Gestion efficace de l'inventaire** : Lorsqu'un agent atteint sa capacité maximale, il retourne directement vers le tanker.
  - **Tanker positionné de manière optimale** : Le tanker se positionne dans un carrefour (au moins trois nœuds atteignables), ce qui évite les blocages et facilite l'accès.
- **Limites** :
  - **Conflits potentiels** : Plusieurs agents peuvent viser le même trésor en même temps, ce qui peut générer des conflits ou des allers-retours inutiles.
  - **Dépendance à la position du tanker** : Si le tanker est inconnu ou mal positionné, cela peut retarder la dépose des ressources et bloquer la collecte.
- **Complexité** :
  - **Temps** : Dépend de la distance entre les agents et les trésors, ainsi que de la fréquence des allers-retours vers le tanker.
  - **Mémoire** : Chaque agent stocke une liste mise à jour des trésors pertinents pour lui, ainsi que sa propre capacité restante.

3.3 Gestion de l'interblocage

- Interblocage entre deux agents (Collect ou Explo) : ils reculent tous les deux de 3 noeuds. Cela marche bien lorsqu'au moins un des deux agents n'a pas encore fini d'explorer, car en reculant, ils recalculent la position suivante, ce qui fait qui change sa direction. Cependant, dans le cas contraire, souvent cela prend du temps car ils reculent tous les deux de 3, puis reviennent, puis recommencent... mais finissent par s'en sortir.
- Interblocage entre un agent Tanker et un agent Collect ou Explo : lorsqu'un agent tombe sur le tanker, il envoie un message "MOVE". Le tanker qui reçoit ce message va décaler d'un noeud puis va y attendre. Lorsque la voie est libre, l'agent s'y place puis continue sa route s'il peut. Il peut y avoir des cas où malencontreusement, le tanker s'est déplacé là où l'agent a voulu aller... C'est ce cas-là qui ne fonctionnent pas correctement dans notre projet. Un autre scénario pourrait être le cas où le tanker est entouré de plusieurs agents, le menant à ne pas pouvoir bouger (tous ses noeuds atteignables sont pris). Nous avons essayé de traiter ce cas en envoyant "BLOCKED" à tous ses voisins. Les agents qui reçoivent ce message, se décale d'un noeud, laissant la possibilité au tanker de bouger.

4 Conclusion et Perspectives d'amélioration

Ce projet nous a permis de mettre en œuvre un système multi-agent coopératif fonctionnel, capable d'explorer un environnement inconnu, de collecter des ressources et de coordonner les déplacements entre agents. Grâce à une architecture modulaire et à l'utilisation de machines à états finis, nous avons pu concevoir des comportements réactifs et adaptés aux contraintes du monde simulé.

Nos agents explorateurs, collecteurs et stockeurs ont su collaborer, en partageant les informations essentielles comme la carte ou la position des trésors, tout en gérant les problèmes de blocage ou de communication partielle.

Pour l'instant, il nous reste encore plusieurs cas à gérer (en incluant l'amélioration des stratégies présentées précédemment), notamment la gestion du Golem. Nous avons réfléchi à ce type de stratégie :
Lorsqu'un agent Explo rencontre un Golem, il le suit. Le but est de prendre les positions atteignables du Golem. Comme il avance aléatoirement, nous avons l'avantage. De la sorte, on peut s'imaginer un cas où le Golem sera bloqué, mais dans ce cas, les agents qui l'entourent aussi. Comme les agents Collect n'ont besoin que de leur liste de trésors pour les collecter, ils n'ont pas besoin des agents Explo pour cela. Si on voulait traiter tous les cas, il faudrait avoir toute la carte puisque le Golem et les agents l'entourant pourraient bloquer le passage des agents Collect, ce qui ralentirait la collecte.