Université Paris Cité

Master 2 - Intelligence Artificielle Distribuée

Module **Planification multi-agents**

Rapport de projet

Collecte multi-agents de trésors par **Abdou Aziz Thiam** & **Yaya Latifou**

Professeure:
Aurélie Beynier

Semester 3 - 2024/2025

Contents

1 Introduction

L'objectif de notre projet était : voir document annexe.

2 Formalisation

Nous avons considéré le projet comme étant un **MMDP** où les agents effectuent leurs actions de manière individuelle tout en ayant un objectif commun : obtenir le meilleur score possible. Nous avons également pris en compte la nécessité de répartir les tâches de manière équitable.

2.1 Modélisation du problème

- Le système est formalisé comme un **MMDP** avec :
  - $N = 6$ agents coopératifs
  - Objectif commun : maximiser un score global
  - Contrainte : équité dans la répartition des tâches

2.2 Description des agents

- **Chest Openers** (Ouvreurs de coffres) :
  - Actions possibles $\mathcal{A}_1 = \{\text{Se déplacer}, \text{Ouvrir un trésor}, \text{Idle}\}$
- **Agents Gold/Stones** (Transporteurs) :
  - Actions possibles $\mathcal{A}_2 = \{\text{Se déplacer}, \text{Charger un trésor}, \text{Décharger un trésor}\}$

2.3 Espace d'états

- Un état $s \in \mathcal{S}$ est défini par :
  - Positions $(x_i, y_i) \in$ Grille pour chaque agent $i \in \{1, ..., 6\}$
  - Positions et statut des trésors (ouverts/fermés, chargés/libres)
- Taille de l'espace : $|\mathcal{S}| = (M \times N)^6 \times K$ (où $M \times N$ = taille de grille, K = configurations de trésors)

2.4 Fonction de transition

- Non-déterministe : $T(s, a, s') = P(s'|s, a_1, ..., a_6)$
- Exemple pour l'action *Se déplacer* :

$$T(s, \text{déplacer}, s') = \begin{cases} 1 & \text{si la cellule cible est libre} \\ 0 & \text{sinon} \end{cases}$$

- Contraintes supplémentaires :
  - Collisions inter-agents bloquantes
  - Actions simultanées résolues par un ordre de priorité prédéfini

2.5 Observations

- Chaque agent a une **observation totale** de l'environnement :
  - Visualisation complète de la grille
  - Positions des autres agents
  - Statut des trésors (ouverts, fermés, chargés, etc.)

2.6 Récompense

- À chaque fois qu'un agent transporteur dépose le trésor au niveau du dépôt, le compteur du trésor s'incrémente.

3 Nos Protocoles

3.1 Policy entre agents ouvreurs (openers)

Nous avons établi la policy suivante pour les agents responsables de l'ouverture des trésors :

- Chaque agent identifie le trésor le plus proche en utilisant l'algorithme A*.
- Les agents communiquent entre eux en envoyant :
  - La position de la case qu'ils souhaitent ouvrir.
  - La distance les séparant de cette case. (voir figure 1.)
- En cas de conflit (deux agents visant la même case) :
  - La mission est attribuée à l'agent ayant la plus petite distance.
  - L'autre agent, écarté de cette mission, se concentre sur sa prochaine cible.
- Le chemin menant à la mission n'est pas figé :
  - L'agent peut ajuster son trajet en temps réel pour contourner des obstacles et atteindre son objectif.
- **État idle (pause)** :
  - Lorsqu'un agent n'a pas de tâche assignée, il se met en pause (idle).
  - L'agent en état idle se place sur une **case vide**, **sans trésor** afin de ne pas gêner la circulation des autres agents.
  - Il reste en pause jusqu'à ce qu'une nouvelle mission lui soit attribuée.

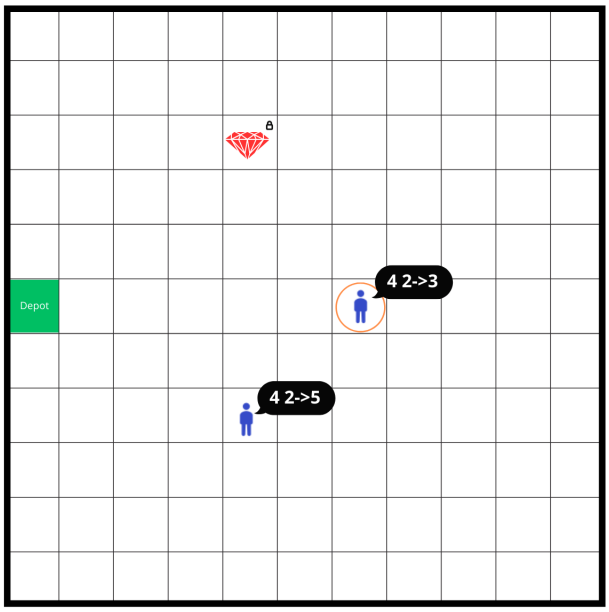

Figure 1: Example de message d'intention

3.2 Policy entre agents transporteurs (Gold ou Stones)

Étant donné que les agents transporteurs effectuent des tâches similaires et répètent les mêmes actions, nous avons établi une politique commune pour les agents **Gold** et **Stones** :

- Chaque agent identifie le trésor le plus proche disponible pour le transport.
- Les agents communiquent entre eux pour :
  - Indiquer quel trésor ils comptent transporter.
  - Partager la distance qui les sépare de ce trésor.
  - Informer de la capacité restante dans leur sac.
- En cas de conflit (deux agents visant le même trésor) :
  - La mission est attribuée à l'agent qui :
    - A la capacité nécessaire pour transporter le trésor.
    - Est le plus proche du trésor parmi ceux qui ont la capacité suffisante.
  - Si aucun agent n'a la capacité de transporter le trésor :
    - Le trésor reste en attente jusqu'à ce qu'un agent libère de l'espace.
  - L'agent écarté de cette mission se redirige vers le prochain trésor compatible avec la capacité restante de son sac.
- Le trajet vers le dépôt n'est pas figé :
  - L'agent peut adapter son chemin en temps réel pour éviter des obstacles ou d'autres agents.
  - En cas de congestion, l'agent peut prioriser un itinéraire plus long mais plus fluide.

3.3 Policy globale à tous les agents

Nous avons suggéré de créer une zone virtuelle appelée **zone dépôt**, définie par un rayon de 2 cellules autour de la zone de dépôt principale. Cette policy vise à optimiser la circulation des agents et à éviter les embouteillages près du dépôt.

- Un agent ne peut entrer dans la **zone dépôt** que si une mission spécifique lui a été assignée dans cette zone (par exemple, déposer, récupérer ou ouvrir un trésor).
- Une fois sa mission accomplie, l'agent doit immédiatement quitter la **zone dépôt** s'il n'a plus d'action à y effectuer.
- Cette restriction permet de :
  - Réduire les embouteillages autour de la zone de dépôt.
  - Assurer une fluidité dans les déplacements des agents, en particulier ceux en transit.
  - Prioriser l'accès aux agents ayant des missions critiques liées au dépôt.
- **Communication entre agents :**
  - Nous n'avons pas jugé nécessaire que les agents s'envoient des messages avant d'aller au **dépôt**.
  - Grâce à la mise en place de la **zone dépôt**, il n'y a pratiquement pas d'attente au niveau du dépôt, ce qui rend la communication supplémentaire inutile dans ce contexte.

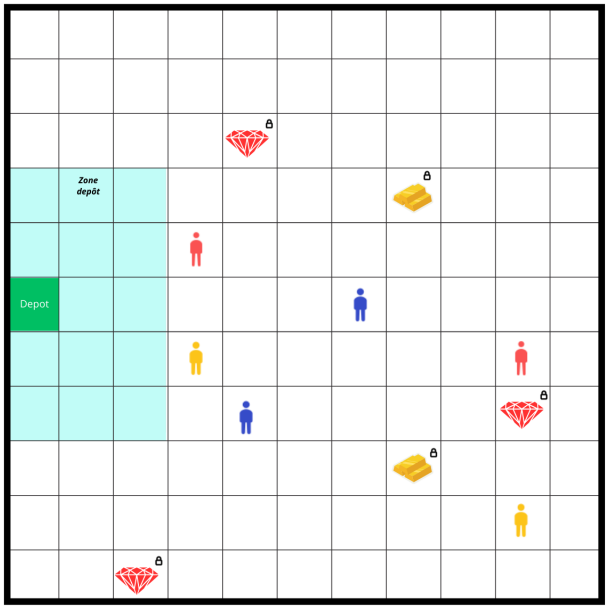

Figure 2: Zone de dépôt

4 Autres

Nous avons suggéré une approche où la grille était virtuellement divisée en deux parties égales, avec un agent de chaque type assigné à une zone spécifique. Cette méthode visait à organiser l'espace de travail des agents pour éviter les conflits et optimiser la répartition des tâches.

Nous avons également implémenté cette approche, mais les résultats se sont avérés moins probants que ceux obtenus avec notre protocole principal. En effet, cette division stricte de la grille conduisait parfois à un déséquilibre dans la charge de travail :

- Certains agents se retrouvaient surchargés (overwork), tandis que d'autres avaient peu de tâches à accomplir.
- La rigidité de la séparation empêchait les agents de s'adapter dynamiquement aux variations de la distribution des trésors.

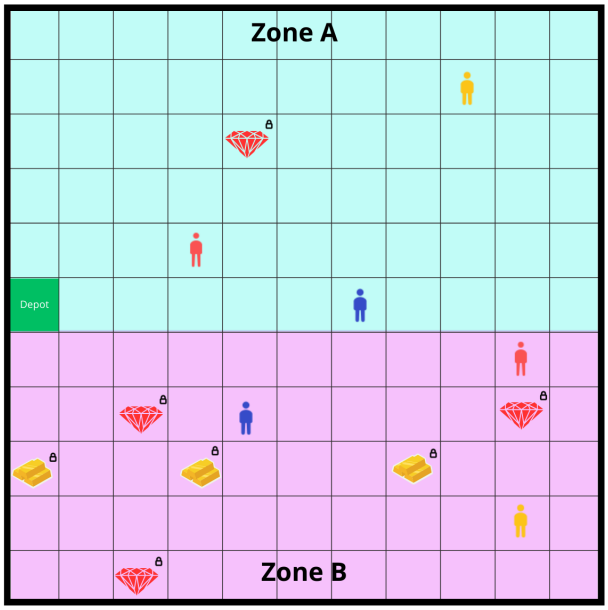

Figure 3: Zone de dépôt

Face à ces résultats, nous avons décidé de ne pas adopter cette méthode et de privilégier notre protocole principal, qui offre une meilleure flexibilité et une répartition plus équilibrée des missions entre les agents.

5 Implémentation

Nous avons implémenté le projet en utilisant **Pygame** pour l'interface graphique, ce qui nous a permis de visualiser en temps réel les déplacements et les actions des agents dans la grille.

- Le jeu dispose d'un **journal de bord** (game log) qui enregistre et affiche les messages échangés entre les agents. Cela permet de suivre la communication et la coordination en temps réel.

- À la fin de chaque partie, il est possible de lancer un **replay**, qui consiste à démarrer une nouvelle simulation avec les mêmes conditions initiales. Cette fonctionnalité permet de rejouer la partie pour observer des variations dans le comportement des agents ou tester de nouvelles stratégies.
- Au niveau du terminal, vous pouvez voir le **score final** ainsi que la **distribution des tâches** entre les différents agents.

Installation des dépendances :

Pour exécuter le projet, il est nécessaire d'installer les bibliothèques requises. Cela peut être fait en utilisant la commande suivante :

```
pip install -r requirements.txt
```

```
python Main.py
```

Le fichier `requirements.txt` contient notamment la bibliothèque **Pygame**, essentielle pour l'interface graphique. Assurez-vous d'avoir `pip` installé sur votre système avant d'exécuter cette commande.

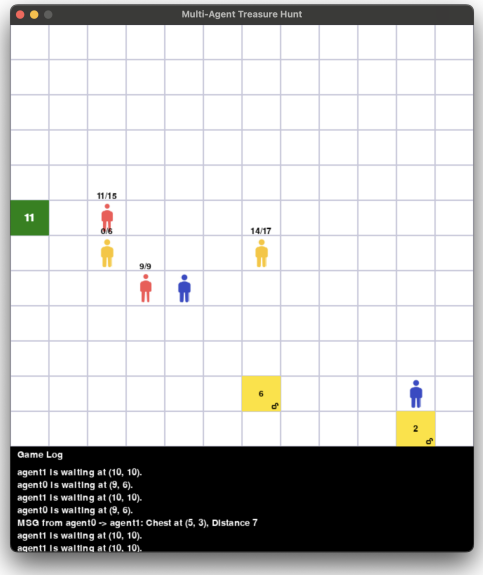

Figure 4: Écran de démarrage du jeu

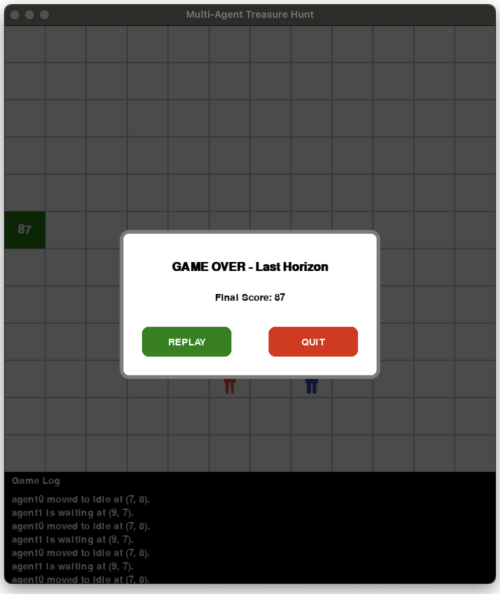

Figure 5: Écran de replay avec les mêmes conditions initiales

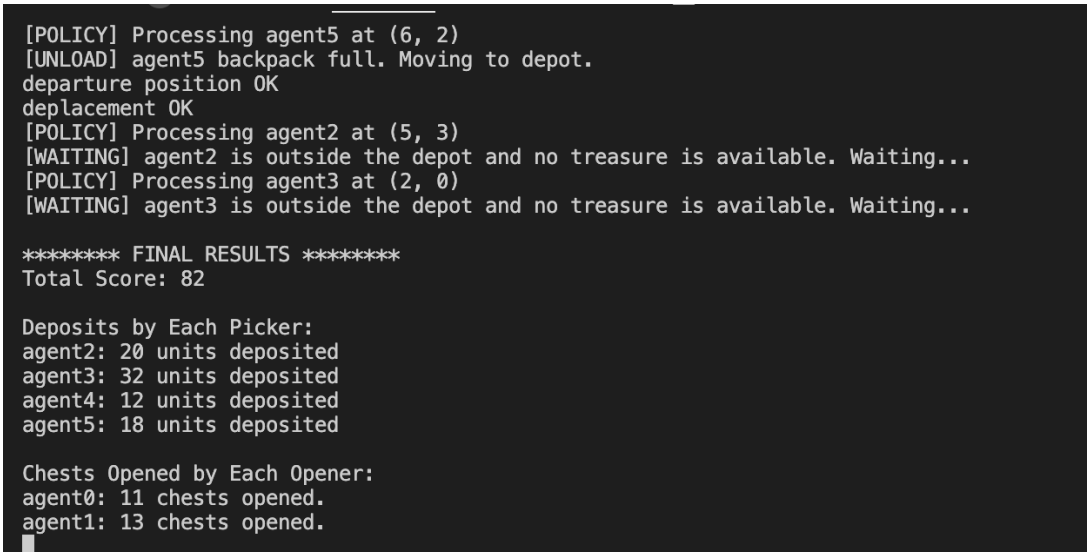

Figure 6: Terminal

6 Tests

Nous avons testé nos politiques dans un environnement différent afin d'évaluer sa robustesse et son adaptabilité. Les résultats ont été très satisfaisants : notre programme fonctionne parfaitement, ce qui démontre que notre code et nos politiques sont capables de s'adapter à un nouvel environnement distribué.

En pièce jointe, nous vous fournissons notre environnement de test. Vous pouvez l'utiliser en copiant son contenu dans le fichier `env.txt`, puis exécuter le programme pour vérifier par vous-même son bon fonctionnement.

7 Conclusion

Ce projet, à la fois challengeant et intéressant, nous a permis de mieux assimiler les notions de **planification multi-agents**.

Grâce à la mise en œuvre des différentes stratégies de coordination et de communication entre agents, nous avons pu explorer des concepts clés tels que la répartition des tâches, la gestion des conflits, et l'optimisation des déplacements dans un environnement partagé.

Cette expérience nous a non seulement permis de renforcer nos compétences techniques en programmation (notamment avec **Pygame**), mais aussi d'approfondir notre compréhension des dynamiques collaboratives dans des systèmes distribués.