Université Paris Cité
UFR Mathématiques et Informatique

Projet Planification Multi-agent

Master 2 Distributed Artificielle Intelligence

Al-Hassane Diallo - Farouk Bennai - Manel Bensalem
Professeur Aurilie Bennier

Année universitaire 2023-2024

Table des matières

1 Introduction

Dans ce rapport, nous explorons une solution au défi de la collecte de trésors dans un environnement multi-agents pour le projet du Master IAD, Le projet consiste à concevoir une solution permettant aux agents d'agir de manière coordonnée et efficace.

Dans un environnement où sont dispersés des trésors de différents types, nécessitant des stratégies de collecte spécifiques. Les agents, dotés de capacités variées et limitées, doivent identifier, accéder, et collecter ces trésors tout en optimisant leurs déplacements et en évitant les collisions. L'enjeu repose sur la dynamique de distribution des trésors et les interactions entre agents de compétences diverses.

Pour atteindre cet objectif, notre approche commence par l'allocation des tâches entre les agents en utilisant une méthode d'enchères. Nous détaillons ensuite notre processus de planification, en commençant par les plans individuels avant d'élaborer un plan global visant à atteindre notre objectif. Enfin, nous exposons notre stratégie pour la résolution des conflits, élément clé pour assurer une coordination efficace entre les agents.

2 Description du problème

On considère des agents devant ramasser la totalité des trésors répartis dans un environnement.

L'environnement est représenté par une grille. Un agent peut se déplacer d'une case à une autre si elles sont adjacentes (les déplacements en diagonale sont permis). Deux agents ne peuvent pas se trouver sur la même case.

Des trésors sont répartis dans l'environnement. Il existe 2 types de trésors : les pierres précieuses et les pièces d'or. Ces trésors sont contenus dans des coffres qui doivent être déverrouillés avant que les agents puissent ramasser les trésors (dès qu'un coffre est déverrouillé, un agent peut ramasser le trésor correspondant sans attendre que tous les coffres soient déverrouillés). Pour ramasser un trésor ou déverrouiller un coffre, un agent ayant la bonne compétence doit se déplacer sur la case du trésor et effectuer l'action voulue.

Les agents ont 3 types différents : les agents ramassant des pièces d'or, les agents ramassant des pierres précieuses et les agents ouvrant les coffres. Un agent ramassant des pièces d'or ne peut pas ramasser des pierres précieuses et réciproquement. Un agent ouvrant les coffres ne peut ramasser aucun trésor.

Chaque trésor peut être ramassé par un seul agent, à condition que le coffre ait été déverrouillé. Une fois ramassé, un trésor doit être déposé dans un point de collecte identifié. Seuls les trésors déposés dans ce point de collecte sont comptabilisés.

Chaque agent dispose d'une capacité de sac à dos qui correspond à la quantité maximum de trésor qu'il peut ramasser. Si la quantité à ramasser est plus grande que la place restante dans le sac à dos, la partie non ramassée du trésor est définitivement perdue. Les agents peuvent toutefois décharger leur sac au point de collecte final pour ensuite collecter d'autres trésors.

3 Allocation de tâches

Dans cette section, nous présentons notre méthode d'allocation des tâches, une étape cruciale dans les systèmes multi-agents. Une allocation des tâches bien conçue est essentielle pour assurer une collaboration harmonieuse et efficace entre les agents, contribuant ainsi à la réussite de la mission collective. Pour notre cas nous utilisons la methode d'allocation de tache des enchéres Anglaise basé sur la distance.

3.1 Méthode des enchéres

La méthode des enchères, dans un contexte général, est un processus par lequel des biens ou des services sont attribués au plus offrant dans un cadre compétitif. Les participants proposent des offres, et l'objet de l'enchère est vendu à celui qui propose le montant le plus élevé.

Dans le cadre des systèmes multi-agents, la méthode des enchères est adaptée pour allouer des tâches ou des ressources parmi un groupe d'agents. Chaque agent soumet une offre basée sur sa capacité à accomplir une tâche ou sa valorisation d'une ressource.

Cette approche permet une allocation dynamique et efficace des tâches, en considérant les compétences et les préférences de chaque agent, ainsi que la maximisation de l'efficience collective. Elle favorise également une compétition saine et équitable, assurant que les ressources sont utilisées de la manière la plus optimale possible par les agents les mieux adaptés.

Dans la section suivante, nous définirons en détail la méthode des enchères anglaises et son application dans le contexte de notre système multi-agents.

3.1.1 Méthode des enchéres Anglaise

La méthode des enchères anglaises est un processus d'enchères ouvert où les participants font des offres successivement plus élevées jusqu'à ce qu'aucune nouvelle offre plus élevée ne soit faite.

Dans le contexte des systèmes multi-agents, cette méthode permet aux agents de surenchérir pour les tâches ou les ressources, comme la collecte de trésors, en augmentant progressivement leurs offres. L'agent avec l'offre la plus élevée remporte l'enchère.

Cette approche spécifique a été choisie pour son efficacité à garantir une allocation optimale des tâches entre les agents, en encourageant une concurrence ouverte et transparente pour l'attribution des trésors, qui va favoriser la maximisation de la valeur récupérée des trésors collectés, en permettant aux agents de surenchérir jusqu'à ce que l'offre la plus avantageuse soit identifiée.

Dans notre système multi-agents pour la collecte de trésors, la méthode d'enchères anglaises est adaptée pour allouer efficacement les trésors dispersés dans l'environnement aux agents.

Une enchère est initiée pour chaque trésor découvert. Les agents soumettent leurs offres en fonction du type de trésor et de leur spécialisation : les agents collecteurs d'or pour les trésors en or, les collecteurs de pierres pour les trésors en pierre, tandis que les agents ouvreurs de coffres peuvent soumettre des offres pour tous les types de tresors.

L'offre d'un agent est déterminée par sa proximité avec le trésor, calculée via une distance euclidienne, et sa capacité de sac, garantissant ainsi que l'enchère favorise ceux qui sont le mieux placés pour effectuer la tâche.

L'agent proposant l'offre la plus avantageuse, c'est-à-dire généralement la plus courte distance, remporte l'enchère et se voit attribuer le trésor à collecter. Cette approche assure une allocation stratégique des trésors, optimisant la collecte tout en minimisant les déplacements inutiles et les chevauchements de tâches entre agents.

La suite du rapport détaillera l'implémentation, la planification et la coordination des agents, ainsi que la gestion des conflits.

3.2 Implémentation

Pour l'implementation de l'allocation des taches avec la methode des encheres Anglaise, nous avons creer une class nommée **TacheAllocation**. Dans cette class nous definissons l'attribut :

— **auctions** : Il s'agit d'un dictionnaire qui stocke les enchères pour chaque trésor. Chaque entrée du dictionnaire a pour clé le nom du trésor et pour valeur un autre dictionnaire contenant les détails de l'enchère, tels que la position du trésor et les offres des agents

Les methodes :

— **start_auction_for_treasure :** Cette méthode initialise une enchère pour un trésor donné avec sa position. Elle prend en paramètres le trésor et sa position. Ce qui permettra aux agents ayant les competences neccessaires de pouvoir surencherie en calculant leur distance avec le tresor et soumettre le offre.

— **bid_on_treasure :** Elle permet de retrouver l'enchère d'un trésor donné ainsi que les offres de chaque agent pour ce trésor. Elle prend en paramètre le trésor et retourne les offres soumises par les agents pour ce trésor.

— **resolve_auctions :**Cette méthode résout les enchères en déterminant l'agent ayant soumis la meilleure offre pour chaque trésor, en fonction de la distance, de la capacité de l'agent, et attribue la tâche de ramassage ou d'ouverture de ce trésor à l'agent qui a remporté l'enchère.

Aprés avoir defini la class **TacheAllocation** ainsi que les methodes permettant de lancer et de resoudre les enchere, nous definissons dans la class des Agents des methodes les permettant d'encherir à des encheres. Pour cela nous definissons les methodes :

— **calculate_distance_to :** Cette méthode permet à l'agent de calculer la distance entre lui et un trésor donné à partir de sa position. Elle utilise la distance euclidienne pour prendre en compte les déplacements en diagonale.

— **evaluate_bid** : Elle évalue la valeur de l'offre à soumettre pour un trésor donné, en se basant généralement sur la distance entre l'agent et le trésor.

— **make_bid** : Cette méthode permet à l'agent de soumettre une offre pour l'enchère d'un trésor donné si sa capacité de sac est suffisante pour contenir le trésor. Elle utilise la méthode **evaluate_bid** pour déterminer la valeur de l'offre.

Aprés avoir defini la class et les methodes neccessaire pour la methode d'allocation de tache des enchere. Nous allons pour chaque trésor trouvé dans l'environnement, une enchère est lancée pour son ramassage ou son ouverture. Les agents surenchérissent en

fonction de leur capacité, calculent la distance jusqu'au trésor et soumettent leur offre. Une fois que tous les agents concernés ont soumis leur offre, les enchères sont résolues et les tâches sont attribuées à chaque agent gagnant. Les tâches attribuées sont stockées dans la variable **bag_contents** de chaque agent. Il faudra noter que un agent ne peut encherir pour un tresor donnés sans avoir la capacité de sac neccessaire ou la competence requise :

- Un agent de type **AgentStone** ne soumet d'offre que pour les tresors de type Stone.
- Un agent de type **AgentGold** ne soumet d'offre que pour les tresors de type Gold.
- Un agent de type **AgentChest** soumet une offre pour tous les tresors.

Voici un exemple des resultats des enchères pour chaque type d'agents (la liste des coffers qu'il a gagné avec les enchères) (pour env1) :

```
agent Chest agent0 (7 , 4)
'treasure' : <Treasure.Treasure object at 0x7f31bebddcf0>, 'position' : (3, 1) 3
'treasure' : <Treasure.Treasure object at 0x7f31beb93c70>, 'position' : (4, 8) 2
'treasure' : <Treasure.Treasure object at 0x7f31beb90880>, 'position' : (6, 3) 6
'treasure' : <Treasure.Treasure object at 0x7f31beb907f0>, 'position' : (7, 0) 2
'treasure' : <Treasure.Treasure object at 0x7f31beb90730>, 'position' : (10, 2) 8
```

```
agent Stone agent3 (5 , 6)
'treasure' : <Treasure.Treasure object at 0x7f31beb908e0>, 'position' : (4, 11) 6
'treasure' : <Treasure.Treasure object at 0x7f31beb90790>, 'position' : (7, 9) 3
```

```
agent Gold agent4 (6 , 7)
'treasure' : <Treasure.Treasure object at 0x7f31bebdd180>, 'position' : (2, 10) 2
'treasure' : <Treasure.Treasure object at 0x7f31beb93c70>, 'position' : (4, 8) 2
```

4 Planification

La planification en système multi-agent concerne l'organisation et la coordination des actions entre plusieurs agents autonomes dans un environnement partagé, pour atteindre des objectifs communs ou individuels. Ces systèmes nécessitent des mécanismes pour gérer la communication, la négociation, et la prise de décisions entre agents, en tenant compte des interactions dynamiques et des interdépendances. La planification multi-agents est utilisée dans divers domaines, comme la robotique, la gestion de trafic, et les simulations sociales, offrant des solutions à des problèmes complexes où plusieurs acteurs doivent opérer de manière coordonnée. Dans ce projet impliquant des agents, la planification joue un rôle crucial pour coordonner les actions et les objectifs des différents agents au sein du système. Voici comment cela pourrait se présenter :

1. Définition des Objectifs :

   Chaque agent ou le système dans son ensemble a des objectifs spécifiques à atteindre, comme trouver et collecter des objets, naviguer vers des emplacements spécifiques, ou optimiser une certaine métrique. Dans notre cas nous definissons un attributs **Goal** dans les classes agents qui contient la liste des objectifs de chaque agent.

2. Connaissance de l'Environnement :

   Les agents doivent avoir une certaine compréhension de leur environnement, qui peut être partagée entre tous les agents (planification centralisée) ou être propre à chaque agent (planification décentralisée pour notre cas).
3. Détermination des Actions :

   Basé sur leurs objectifs et leur connaissance de l'environnement, les agents déterminent les actions nécessaires pour atteindre leurs objectifs. Cela peut impliquer des algorithmes de recherche de chemin, des stratégies d'évitement d'obstacles, ou des méthodes de prise de décision basées sur des règles.
4. Résolution de Conflits :

   Dans un environnement multi-agents, les actions d'un agent peuvent affecter ou entrer en conflit avec celles d'un autre. La planification doit donc inclure des mécanismes pour détecter et résoudre ces conflits, par exemple, en réassignant des objectifs, en modifiant les plans pour éviter les collisions, ou en instaurant des priorités entre les agents.
5. Adaptabilité :

   Les agents doivent être capables d'adapter leurs plans en réponse aux changements dans l'environnement ou aux actions des autres agents. Cela peut impliquer une planification réactive ou des mécanismes de replanification.
6. Optimisation :

   Enfin, la planification peut chercher à optimiser certains critères, comme minimiser le temps total pour accomplir tous les objectifs, réduire la distance parcourue par les agents, ou maximiser l'efficacité globale du système.

Pour appliquer ces principes à notre projet, nous allons commencer par définir clairement les objectifs de chaque agent et du système dans son ensemble. Ensuite, concevoir la logique de planification qui permet aux agents de déterminer et d'exécuter les actions nécessaires pour atteindre leur objectifs, tout en intégrant la gestion des interactions et des conflits entre agents. La flexibilité et l'adaptabilité seront essentielles pour gérer les dynamiques complexes d'un environnement multi-agents.

4.1 Protocole PGP

Le Protocole PGP, dans le contexte de notre projet multi-agents, désigne le mécanisme par lequel les agents coordonnent leurs actions pour former un plan global efficace, tout en conservant la flexibilité de s'adapter aux changements dynamiques de l'environnement.

Cette approche s'appuie sur l'élaboration de plans individuels qui prennent en compte les objectifs personnels de chaque agent ainsi que la mission globale. Le Protocole PGP permet une intégration des plans individuels en un cadre cohérent, assurant que chaque agent contribue au succès collectif sans entraver les actions des autres.

Ce processus implique une communication régulière entre les agents pour partager les mises à jour de statut, les décisions prises et les ajustements de planification nécessaires. L'objectif est de garantir que tous les agents avancent vers les objectifs communs de

manière synchronisée et efficace, en minimisant les conflits et en optimisant la répartition des tâches et la collecte des trésors.

4.2 Plans individuels

La planification individuelle concerne la manière dont chaque agent crée un plan pour atteindre ses objectifs de manière autonome, en tenant compte de l'environnement dans lequel il évolue et des actions potentielles des autres agents.

Une stratégie commune pour la planification individuelle est l'utilisation d'algorithmes de recherche et d'optimisation qui permettent à chaque agent de déterminer la meilleure séquence d'actions pour atteindre ses objectifs. Ces algorithmes peuvent inclure des méthodes de recherche heuristique, telles que l'algorithme A* ou des algorithmes évolutionnaires.

Une planification individuelle efficace dans un contexte multi-agents nécessite donc un équilibre entre l'autonomie individuelle et la coopération, tout en étant robuste aux imprévus et dynamique pour répondre aux changements.

Dans le contexte de notre projet multi-agents pour la collecte de trésors, les plans individuels des agents sont établis en utilisant l'algorithme A*. Chaque agent calcule son propre chemin vers les tresors, en prenant en compte leur type spécifique, pour identifier le tresor le plus proche à ouvrir ou à ramasser.

Agent "chest" : L'agent dédié à l'ouverture des coffres, élabore un plan individuel qui trace son itinéraire vers tous les coffres qu'il doit ouvrir, en se basant sur les résultats des enchères pour prioriser son chemin du plus proche au plus éloigné.

Les agents "gold" et "stone" : De manière similaire, les agents collecteurs d'or et de pierres construisent leurs plans individuels pour naviguer vers les trésors qu'ils ont remportés lors des enchères, et qui leur son affectés en utilisant également l'algorithme A* pour déterminer l'itinéraire le plus court vers le tresor le plus proche par rapport à sa position.

Ainsi, chaque agent, qu'il soit spécialisé dans l'or, les pierres, ou l'ouverture de coffres, se voit attribuer une liste de chemins vers des tresors dont il est responsable, optimisant la collecte de trésors dans l'ensemble du système.

Voici un exemple des plans individuels pour chaque agents (pour env1) :

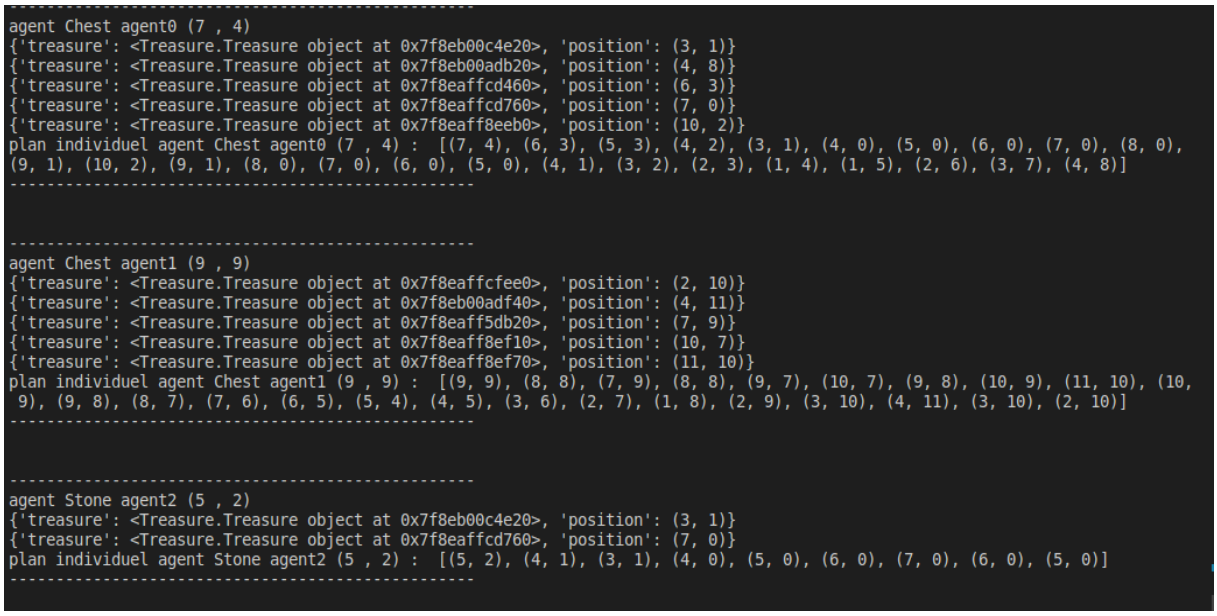

FIGURE 1 – Plan Individuel

On peut voir sur cette image l'ensemble des taches qui sont allouées a un agent ainsi que le plan qu'il a etablis pour atteindre ces objectifs.

4.3 Plan global et Resolution de conflits

Dans notre projet où il y'a la coordination entre agents, la construction du plan global se fait de manière collaborative et continue. Chaque agent communique avec les autres en partageant régulièrement son plan. Une fois qu'ils ont reçu les plans des autres agents, ils analysent localement s'il existe des conflits potentiels, ce qui pourrait se produire si plusieurs agents prévoient d'occuper la même position au même moment.
Nous avons definis pour chaque type d'agent un attribut **priority** qui permet de donner une priorité de passage entre les agents. Et comme les agents ouvreur doivent ouvrir les coffres afin que les agents ramasseurs puissent venir recuperer le tresor alors ils ont la plus grande valeur de priorité.
Ce qui fait que en cas de conflit détecté, l'agent envoie un message à l'autre agent concerné, contenant des informations sur sa priorité et la distance restante jusqu'à son objectif. L'agent destinataire compare alors cette priorité avec la sienne. S'il a une priorité plus basse, il ajuste son plan en conséquence et le renvoie à tous les agents. Si la priorité est égale, l'agent compare les distances restantes jusqu'à l'objectif, ajustant son plan si sa distance est plus grande. Si la distance est égale ou plus petite, un message est renvoyé à l'agent initial pour qu'il ajuste son plan.

Ce processus de détection des conflits, de négociation et de replanification se poursuit de manière itérative jusqu'à ce que tous les agents parviennent à un consensus sur un plan global sans conflits. La communication continue entre les agents, combinée à la capacité de modifier et de partager leurs plans, garantit une coordination efficace dans notre environnement multi-agent.

La methode **detecter_et_negocier_conflits** dans la class **MyAgent** permet à chaque agent de partager son plan et de recevoir les plans des autres agents.

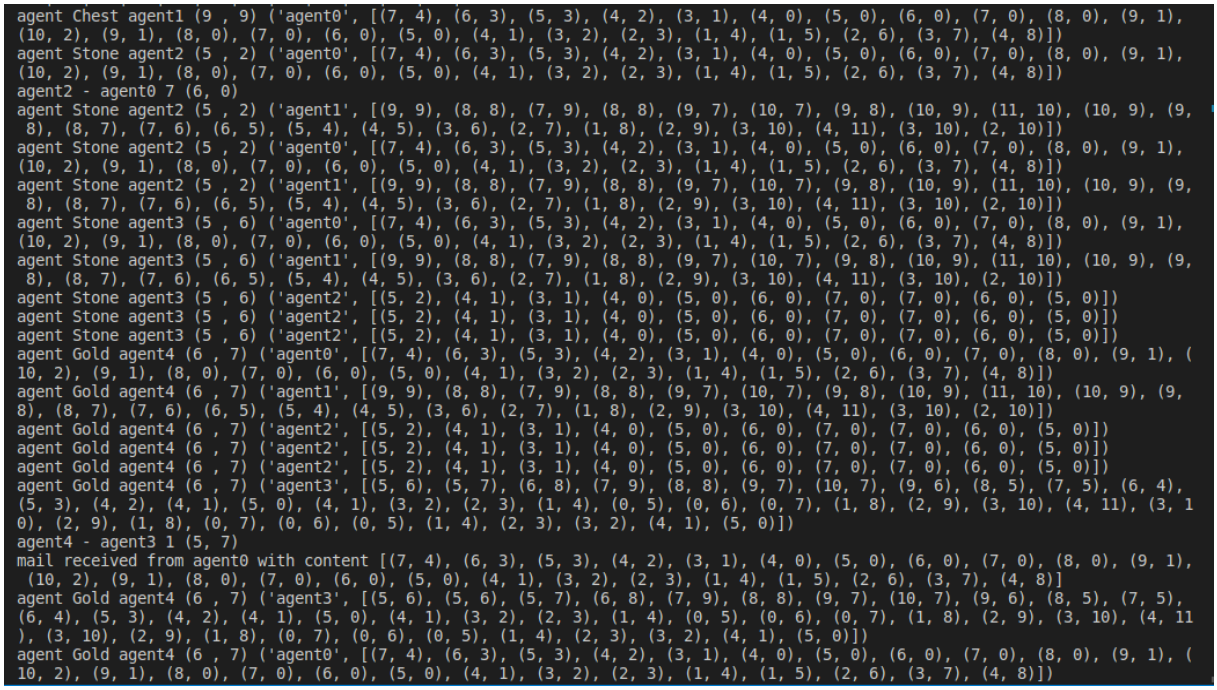

FIGURE 2 – Exemple d'echange de node Plan

4.4 Implementation

La fonction **a_star_search** dans la classe **TacheAllocation** met en œuvre l'algorithme A* pour trouver le chemin le plus court entre un agent et son but(un trésor ou le point de collecte). Elle utilise une grille représentant l'environnement, où les obstacles et les chemins possibles sont définis. L'algorithme évalue les coûts des chemins potentiels en combinant le coût réel pour atteindre un nœud avec une estimation heuristique de la distance restante jusqu'à l'objectif. Cette fonction priorise les nœuds avec le coût total le plus bas, permettant ainsi aux agents de naviguer efficacement vers leur cible tout en contournant les obstacles.

La méthode **planindividuel** dans la classe **MyAgentChest** est conçue pour créer un plan d'action individuel pour l'agent, lui permettant de naviguer vers et d'ouvrir les coffres qu'il a gagnés dans les enchères. Cette méthode commence par localiser le trésor le plus proche en utilisant l'algorithme A* pour chaque trésor attribué. L'agent planifie ensuite un itinéraire optimal vers ce trésor, met à jour sa position après avoir atteint chaque trésor, et répète le processus jusqu'à ce que tous les coffres de trésors qui lui sont attribués soient ouvert. Ce processus illustre une approche méthodique pour maximiser l'efficacité de la collecte des trésors en minimisant les déplacements inutiles.

La méthode **planindividuel** dans les classes **MyAgentGold et MyAgentStones** permet à l'agent de construire un plan d'action pour collecter des trésors d'or ou de pierres. L'agent évalue d'abord quels sont les trésors qu'il peut ramasser en fonction de la capacité restante dans son sac. Si le sac est plein ou si le trésor suivant ne peut pas être ramassé, l'agent planifie un retour au point de collecte pour décharger le contenu de son sac et ensuite retourner ramasser les tresors qui lui reste à ramasser sinon s'il a executer tous

son plan alors on supprime l'agent de l'environnement pour vider la grille où il se trouvait. Cette méthode utilise l'algorithme A* pour déterminer le chemin le plus court vers le trésor le plus proche ou vers le point de collecte, optimisant ainsi les déplacements de l'agent et maximisant la quantité de tresor collecté avant de devoir retourner décharger.

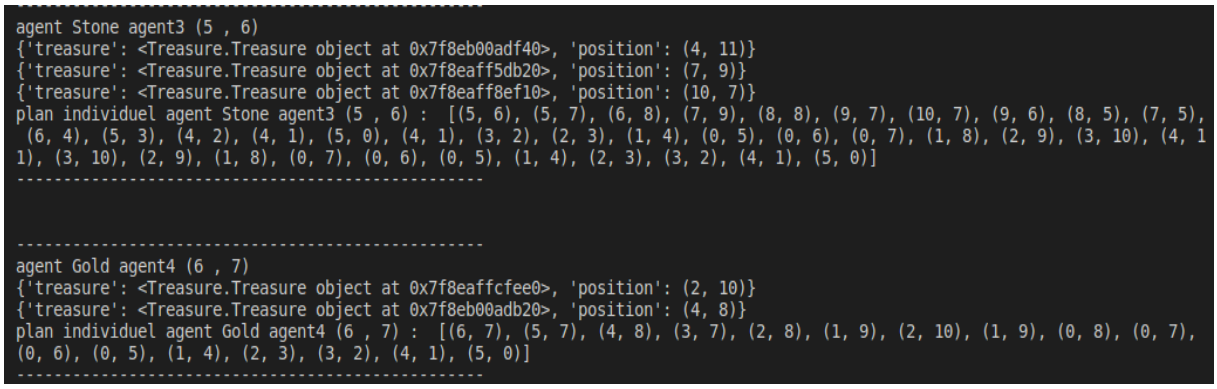

FIGURE 3 – Potentiel conflit entre agent3 et agent4 au step 1

Un exemple de potentiel conflit entre les agent3 et agent4 au step 1. Dans les plans plan individuel on voit que les agent3 et agent4 peuvent se retrouver dans la méme case à l'instant t=1, dans ce cas de figure il faut mettre en place la politique de gestion des conflits. Pour la gestion de ce conflit l'agent3 envoie un node plan à l'agent4 qui contient le step t=1,la distance par rapport à son objectif et la position du conflit (5,7). exemple node plan (1,1,(5,7)).

Aprés avoir lancé et resolu les enchères pour la repartitions des taches et la planification individuelle pour chaque agent, on lance la methode de construction du plan globale qui consiste à detecter en premier les potentiels conflits entre les plan individuels des agents et ensuite lance la resolution de ces conflits.

5 Execution

Dans cette partie nous allons executer notre code et voir les resulats. Pour cela il faut etre dans le repertoire **projetDAI** et lancé la commande **python3 Main.py** Dans le fichier Main.py on trouve :

- **tache=TacheAllocation(env)** :
  tache.start_auction_for_treasure(env.grilleTres[i][j],i,j) pour lancer un enchére pour un tresor avec sa position. Pour la partie Allocation de Tache
- **a.planindividuel(tache)** : pour la construction du plan individuel d'un agent
- **a.detecter_et_negocier_conflits(tache)** : pour commencer la communication et l'echange de plan avec les autres agents
- **Planification(env=env).executonPlanGlobal()** : Pour l'execcution du plan global aprés resolution des conflits

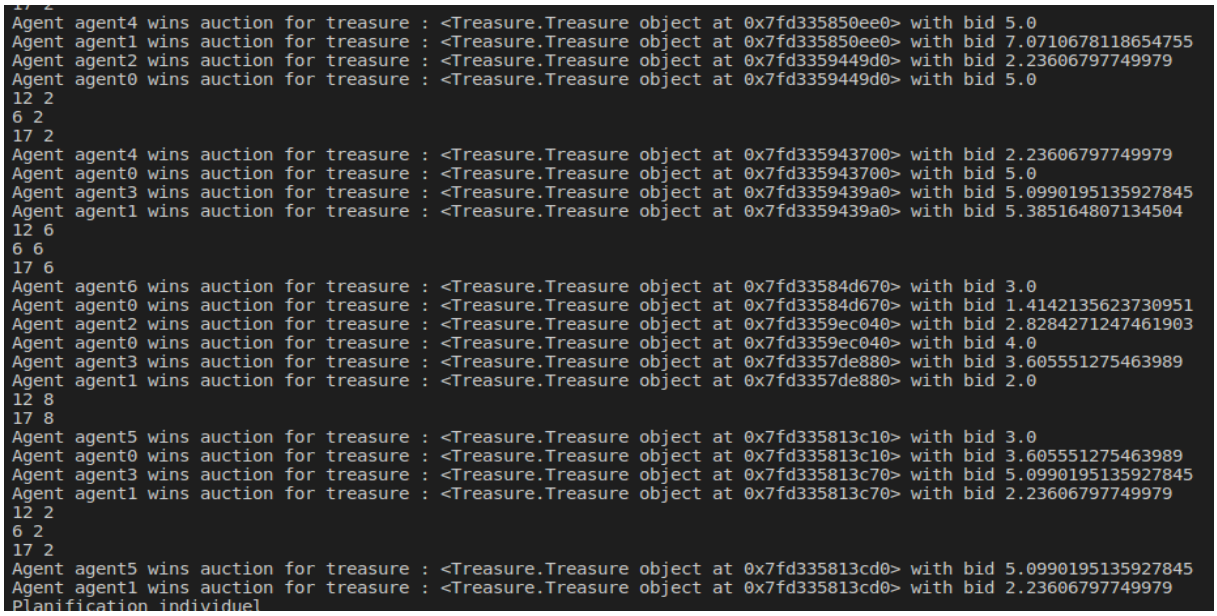

FIGURE 4 – Resultat des Enchéres

Sur cette image on peut voir tous les resultats des encheres. On peut voir le trésor et l'agent qui à remporté l'enchère.

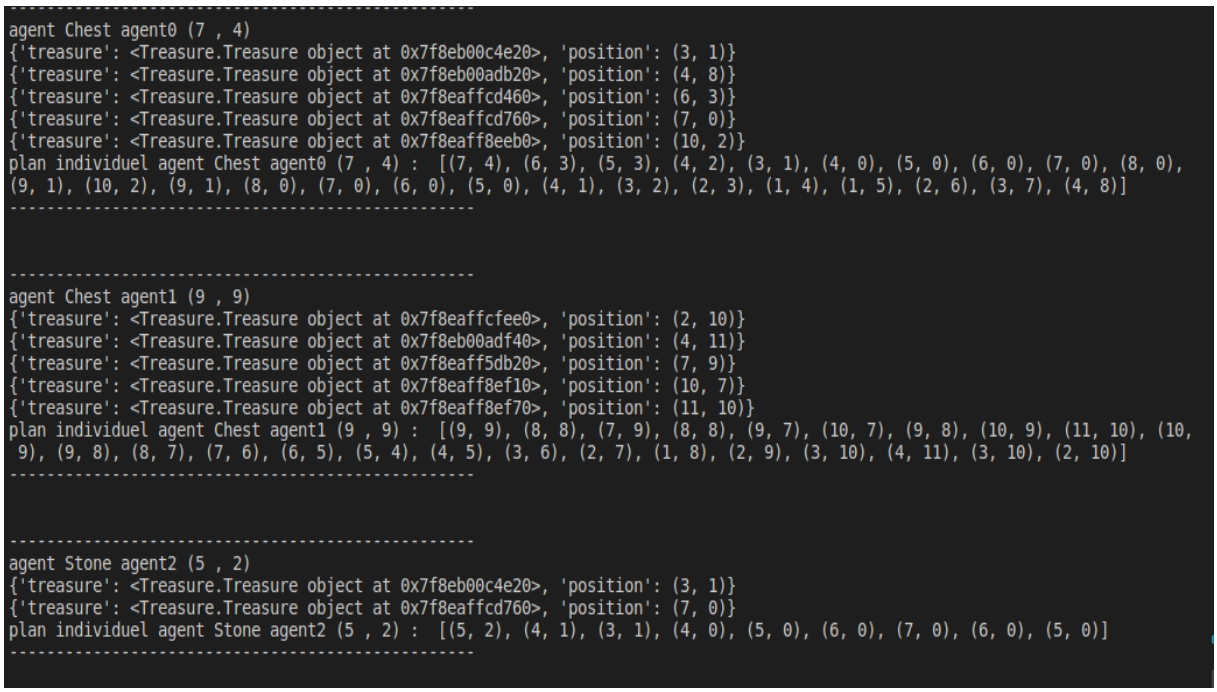

FIGURE 5 – Plans individuel avant resolution de conflit

Sur cette figure on peut voir l'ensembles des taches qui sont attribuées à un agent(le contenu de sont bag_content) et le plan qu'il a etabli pour atteindre ces objectifs.

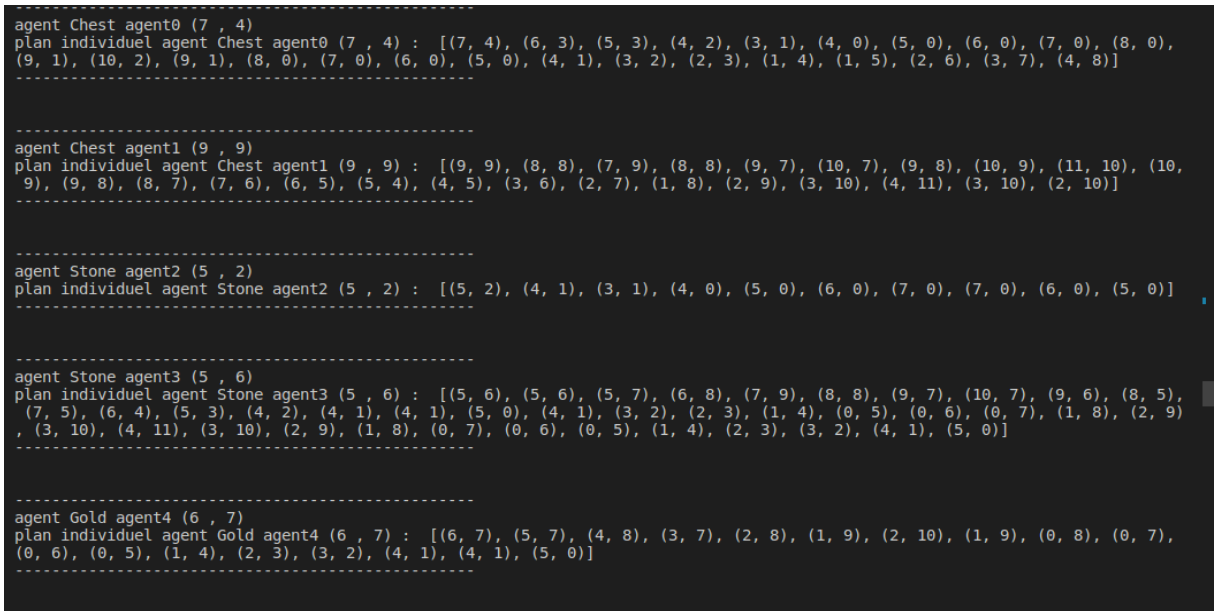

FIGURE 6 – Plan Individuel aprés resolution des conflits

Il suffit de comparer les deux images c'est à dire regarder les plans de agent pour voir la resolution de des conflits. Par exemple au step t=1, les agent3 et agent4 avait un conflit à la position (5,7) et aprés la resolution on voit que l'agent 3 etant plus loin de son but doit faire une action wait c'est à dire ne bouge pas il reste sur la méme case jusqu'à ce que l'agent4 passe.

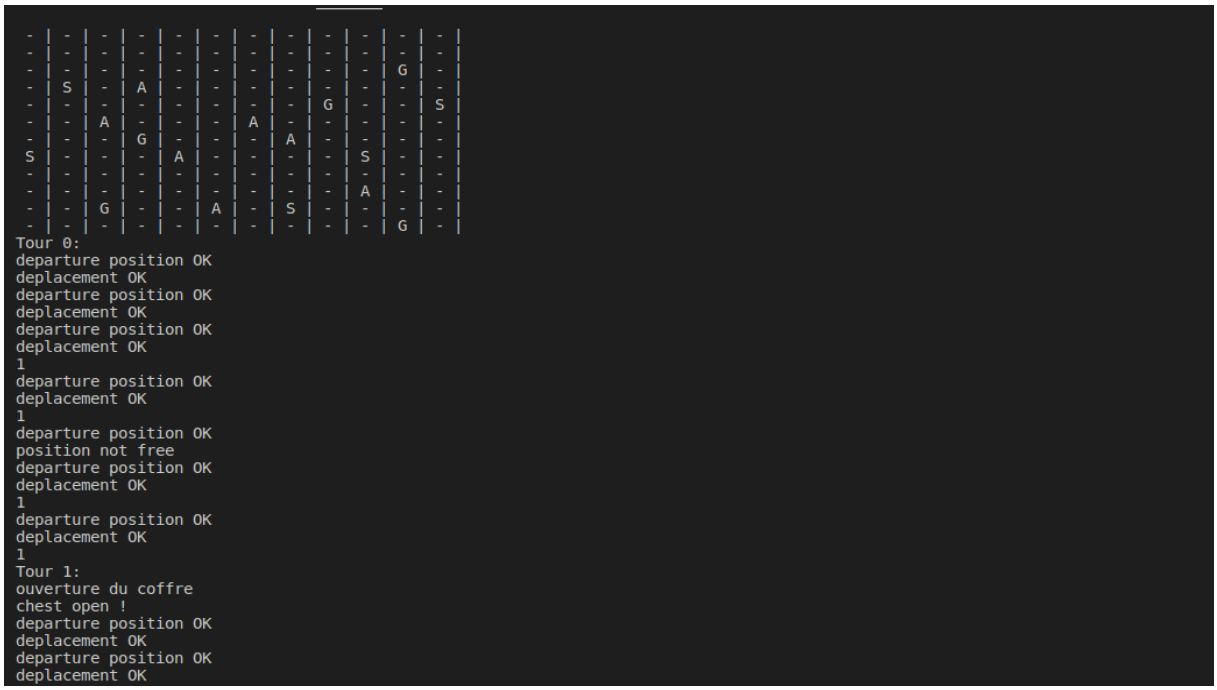

FIGURE 7 – Execution du plan Global

Cette capture d'ecran montre l'excution du code en motrant les differents deplacements des agents pour le ramassage des tresors ainsi que le score c'est à dire la quantité de tresor ramasser l'execution se fait en appelant la méthode **Planification(env=env). executonPlanGlobal()**.

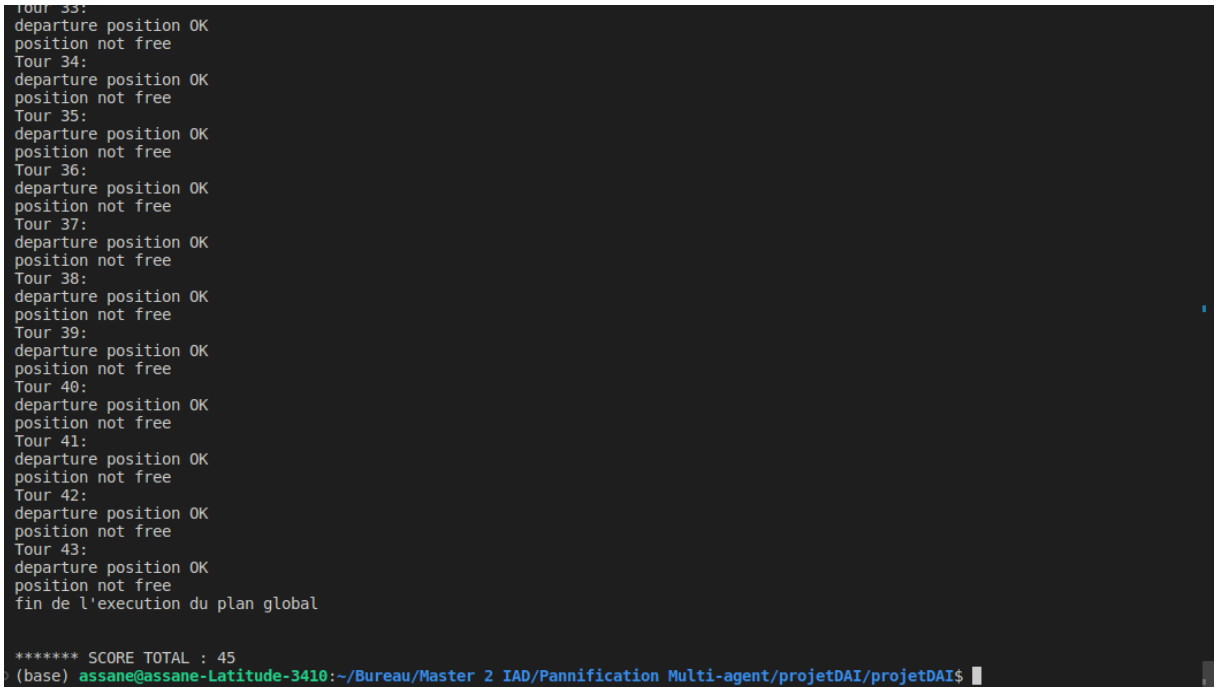

FIGURE 8 – Execution du plan Global avec le ficheir env1.txt

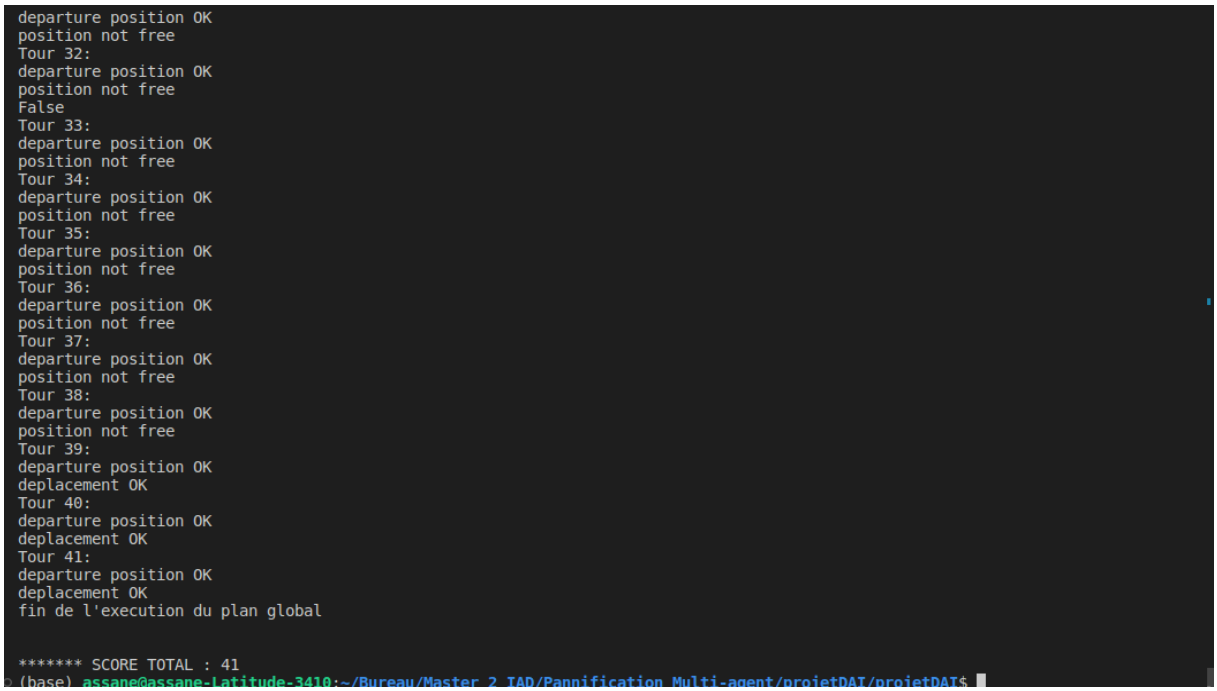

FIGURE 9 – Execution du plan Global avec le ficheir env2.txt

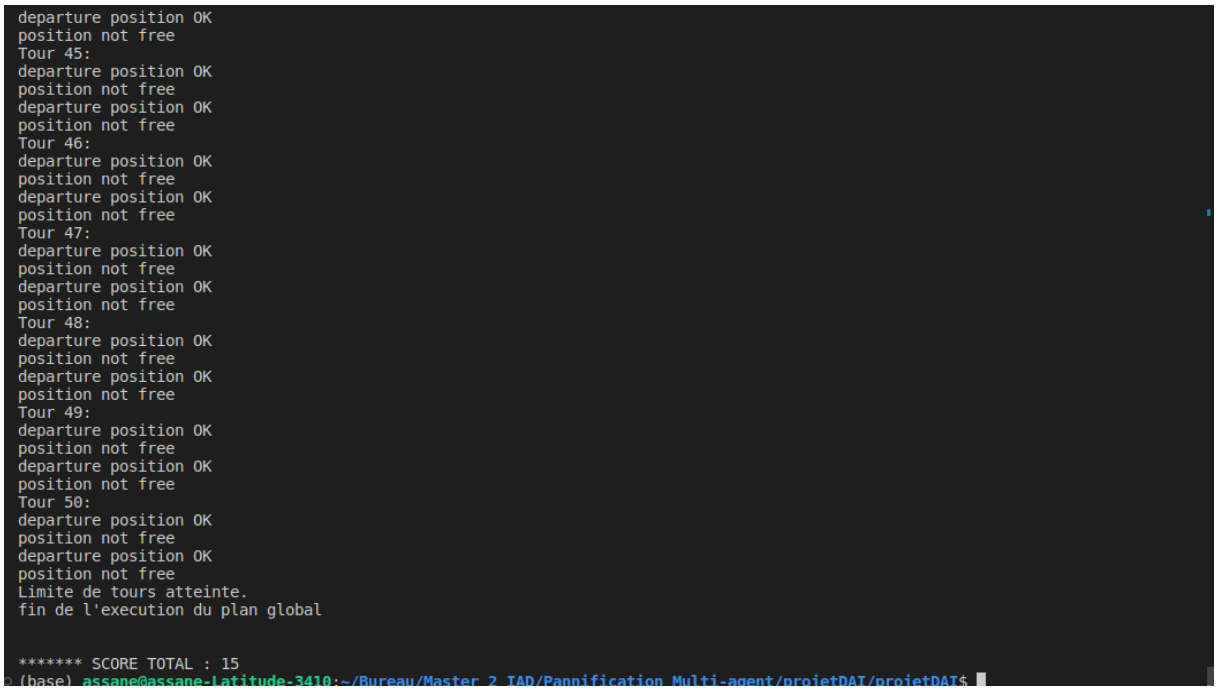

FIGURE 10 – Execution du plan Global avec le ficheir env3.txt

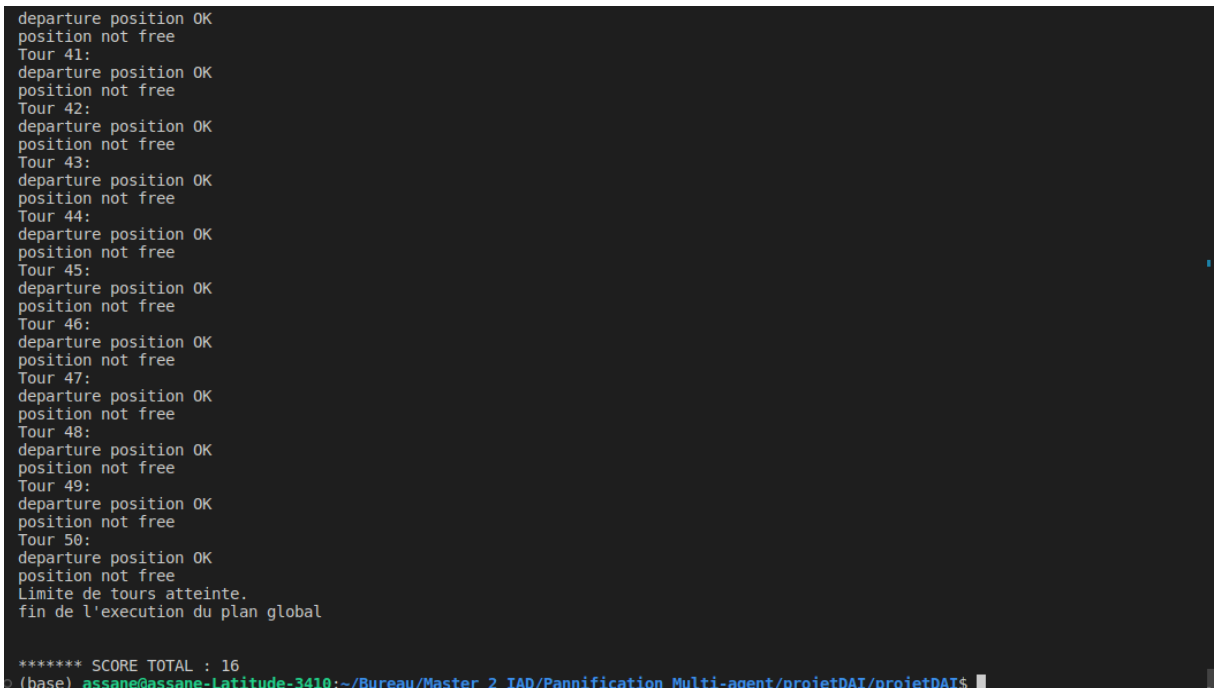

FIGURE 11 – Execution du plan Global avec le ficheir env4.txt

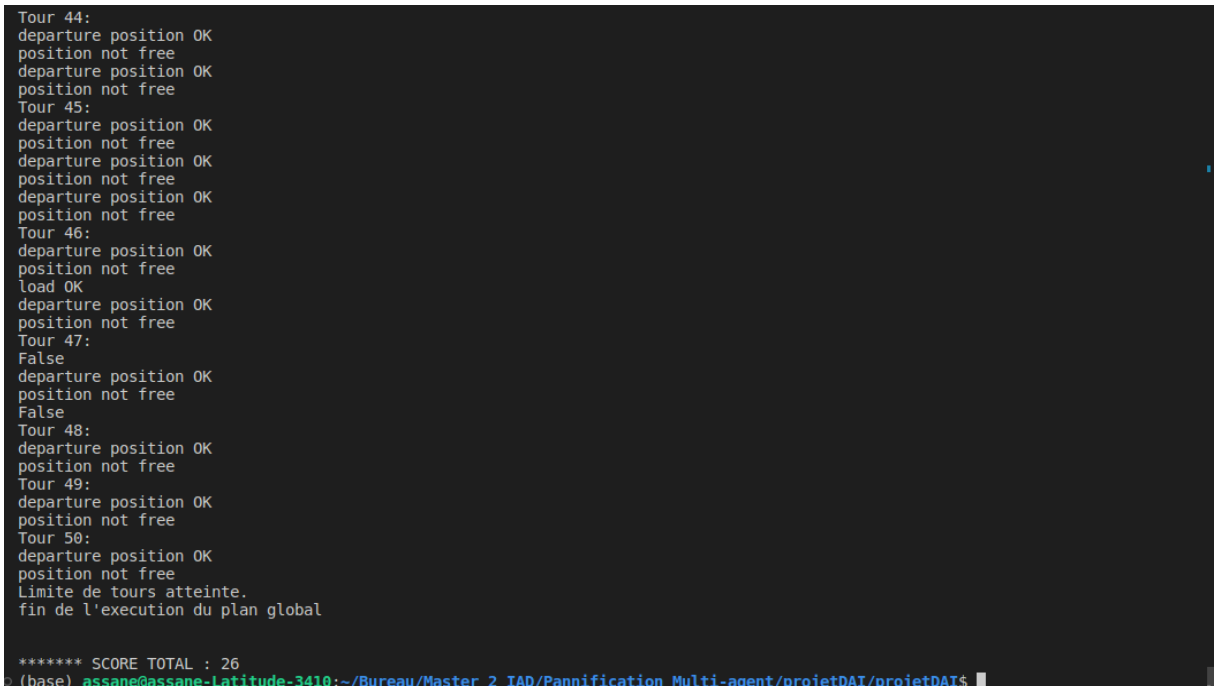

FIGURE 12 – Execution du plan Global avec le ficheir env5.txt

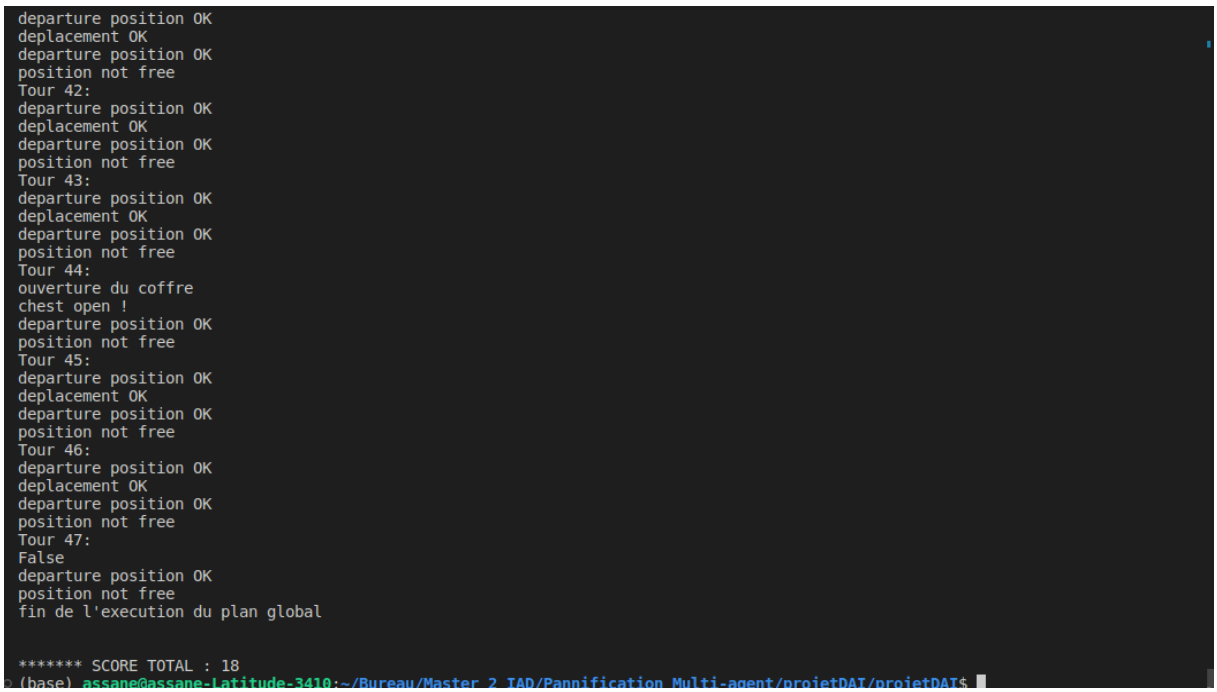

FIGURE 13 – Execution du plan Global avec le ficheir env6.txt

6 Conclusion

En conclusion de notre projet sur la planification multi-agents pour la collecte de trésors, nous avons développé et mis en œuvre une méthode d'enchères anglaises pour l'allocation des tâches et utilisé l'algorithme A* pour la planification des itinéraires individuels.

Cette approche a permis une coordination efficace entre les agents, optimisant la collecte des trésors tout en minimisant les conflits et les déplacements inutiles. Nous avons lancé notre programme sur differentes configuration d'environnement et les résultats démontrent l'efficacité de notre système dans la gestion des tâches complexes dans un environnement multi-agents, soulignant l'importance d'une planification stratégique et d'une allocation des tâches bien conçue.

Ce projet illustre le potentiel des méthodes de planification distribuée dans la résolution de problèmes réels et complexes, offrant des perspectives prometteuses pour de futures recherches et applications dans le domaine de l'intelligence artificielle distribuée.

Informations utiles

Si vous souhaitez créer des instances d'environnement, vous pouvez éxecuter le fichier **GenerateEnv.py** en specifiant la taille du grid, le nombre de trésors, le nombre d'agents et un fichier **EnvNEntier.txt** sera crée dans le méme repertoire. Autre precision, si vous n'arrivez pas à éxecuter le fichier mettez vous en mode Admin pour l'éxecuter. Aller dans le fichier et utilisez la syntaxe suivante :
Environment(TailleX,TailleY, NombreDeTresors, NombreAgent)
Exemple :**Environment(30, 30, 15, 10)**

Pour une meilleur experimentation, utilisez l'instance d'environnement du fichier : **env10.txt**.
Lancer le fichier **Main.py** pour l'execution du programme avec la commande **python3 Main.py** dans le repertoire du projet.