Programmation orientée agents.

AISSAOUI Lucas

21 décembre 2021

Table des matières

1 Introduction

Ce rapport rend compte du Projet jeu de conception et implémentation d'un système multi-agents durant l'unité d'enseignement "HAI716I - Programmation orientée agents" enseignée par madame CROITORU Madalina. L'ensemble des fichiers de codes est fourni dans le dossier joint avec ce document et sont exécutables avec python.

2 Modélisation

2.1 Modélisation des agents

Pour modéliser un système multi-agents, nous avons déjà besoin d'agents. Un agent peut être définit de la manière suivante :

"*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.*"

Cette citation issu du livre "Artificial Intelligence A Modern Approach" de Stuart Russell et Peter Norvig nous permet de définir de manière simple et claire ce qu'est un agent. Dans notre implémentation, notre agent évoluera dans un plan à deux dimensions avec huit actions possibles pour se déplacer et disposera de méthode pour savoir si il est possible d'atteindre certaines positions, car l'environnement sera composé d'obstacles et d'espaces libres. La structure de notre environnement sera détaillé dans la partie suivante. Nous allons donc utiliser cette représentation UML pour notre agent :

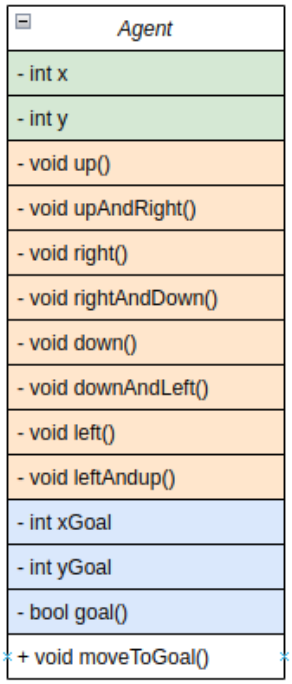

FIGURE 1 – Diagramme UML de la classe agent.

Dans notre, classe x et y représentent la position de notre agent dans l'espace, c'est ce qui va représenter son état dans notre modélisation. Les huit procédures suivantes représentent les actions à sa disposition pour se déplacer. On considère que notre agent connaît son but, cette connaissance est représentées par xGoal et yGoal qui sont les coordonnées de son but. La méthode moveToGoal est la seule méthode publique de notre agent et c'est elle qui va contenir la stratégie d'exploration.

2.2 Modélisation de l'environnement

Comme vu précédemment, l'environnement de nos agents est un plan en deux dimensions représenté par une matrice à deux dimensions construite à partir d'une image au format PNG. Le constructeur de notre classe va d'abord convertir l'image en noir et blanc pour créer une matrice de même taille que l'image pour que chaque pixel de l'image corresponde à un élément de notre matrice. Notre matrice contiendra des 1 pour les murs et des 0 pour les espaces libres. Pour l'affichage, nous utiliserons Tkinter pour sa simplicité d'usage et sa polyvalence.

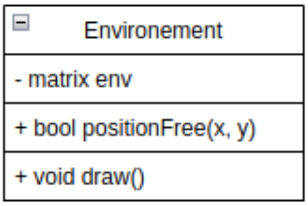

FIGURE 2 – Diagramme UML de la classe environnement.

Cette classe, aussi simple qu'elle puisse paraître, possède tout le nécessaire pour jouer le rôle d'environnement pour nos agents. La variable "env" est une matrice contenant les valeurs qui représentent les murs et les espaces libres, la méthode suivante permet à un agent de savoir si une position (x, y) est libre ou non. Une méthode "draw" nous permet d'afficher l'environnement à l'écran.

2.3 Modélisation du système multi-agents

Maintenant que nos classes agent et environnement sont fixées, il nous faut mettre en commun les deux. Pour cela, nous avons implémenté une classe SMA qui regroupera un ensemble d'agents représenté par un tableau d'agents et une instance de notre classe environnement précédemment créée. Notre classe SMA suivra le diagramme UML suivant :

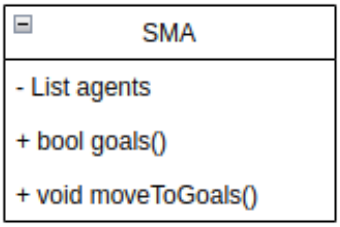

FIGURE 3 – Diagramme UML de la classe SMA.

La variable "agents" est une liste d'agents ayant tous le même but. La fonction "goals" nous permet de savoir si tous nos agents ont atteint leur but. Pour finir la méthode moveToGoals appelle la méthode moveToGoal pour chacun de nos agents. Maintenant que notre écosystème est en place, nous pouvons mettre en place notre stratégie d'exploration.

3 Exploration de l'environnement

3.1 Stratégie d'exploration et modèle BDI

Pour notre stratégie d'exploration, nous allons utiliser une combinaison de deux algorithmes. Dans un premier temps, notre agent va sélectionner parmi les huit actions disponibles une liste d'actions possibles qui mène à une position non explorée. Notre agent va ensuite choisir l'action qui le rapproche le plus de son but, on utilise pour heuristique la distance euclidienne entre une position et son but. Une fois cette sélection faite, notre agent effectue l'action correspondante. Dans certains cas, notre agent peut se retrouver dans un état où la liste des actions possibles et non explorées est vide. Dans ce cas, notre agent se considère comme étant bloqué et va exécuter l'algorithme de Pledge pour sortir de cette position tant que la liste des actions possibles et non explorées est vide. Cet algorithme permet de sortir d'un labyrinthe à coup sûr si il n'y a pas d'îlots dans le labyrinthe. Le principe de cette solution est de longer un mur par la droite (ou la gauche) tant que l'agent n'est pas sorti. Notre stratégie de recherche est donc une combinaison d'une recherche gloutonne avec pour heuristique une distance à vol d'oiseau et un algorithme de Pledge pour sortir des impasses. Cette stratégie nécessite de mémoriser les espaces visités, pour cela s'offrent à nous deux solutions, soit nos agents ont chacun leur propre mémoire ou bien ils peuvent partager leur connaissance dans une seule et unique commune mémoire. Nous étudierons les deux méthodes dans la suite de ce document.

Le modèle BDI (Beliefs-Desires-Intentions) est l'un des modèles d'architecture les plus classiques. Il est composé de trois parties :

- Beliefs : L'ensemble des positions libres autour de notre agent.
- Desires : L'ensemble des positions non explorées.
- Intentions : L'action qui mène à la meilleure position libre non explorée

Intentions peut être vu comme l'intersection de Beliefs et Desires, cet ensemble peut contenir plusieurs actions mais dans notre cas, seule la meilleure sera retenue par notre agent. Dans certains cas, l'ensemble d'actions Intentions peut s'avérer être vide (situation de blocage) et c'est pour cela que nous avons utilisé l'algorithme de Pledge.

3.2 Stratégie de recherche sans partage de mémoire

Dans cette stratégie, on considère que chaque agent a son propre "cerveau", il est le seul à connaître les espaces qu'il a déjà exploré. Certes, un système multi-agent sans communication entre agent n'est pas vraiment un système multi-agents, mais cette étude nous permettra de fixer un point de comparaison pour estimer les gains potentiel (ou non) du partage des informations entre agents. Nous verrons par la suite les inconvénients et les avantages.

3.3 Stratégie de recherche avec partage de mémoire

Dans cette stratégie, on considère que chaque agent lit et écrit dans un seul et unique espace de mémoire partagé, cette fonctionnalité nécessite de modifier légèrement les classes vues précédemment. Les modifications au niveau du code source sont considérées comme négligeables et ne seront pas détaillées dans cette partie. La mémoire partagée sera stockée et gérée par l'environnement qui sera enrichi de méthodes pour permettre aux agents d'écrire et lire dans cette matrice. Pour plus de détails, le code source est disponible à l'adresse suivante : AissAiss/SMA

Mes modifications du solveur par défaut n'ont pas abouti et je n'ai pas trouvé de résultat concluant pour améliorer le solveur par défaut de choco qui reste relativement efficace pour ce genre de tâche.

4 Expérimentation et premiers résultats

4.1 Méthodologie d'expérimentation

Nous avons décidé d'utiliser des labyrinthes pour tester nos deux stratégies. Les figures qui vont suivre seront des images des résultats d'exploration menés par des systèmes de 4 agents, à droite on pourra voir en couleur les emplacements explorés par nos agents, ceci nous permettra de mieux visualiser les choix et parcours faits par nos agents.

Pour commencer, nous avons mis en place une configuration de départ très basique, puis nous avons exécuté nos deux stratégies l'une après l'autre, on compare ensuite le nombre d'actions de chaque agent pour atteindre leur but et enfin on fait une somme pour connaître le nombre total d'espaces explorés ; ainsi on peut déterminer la meilleure des deux stratégies dans cette configuration.

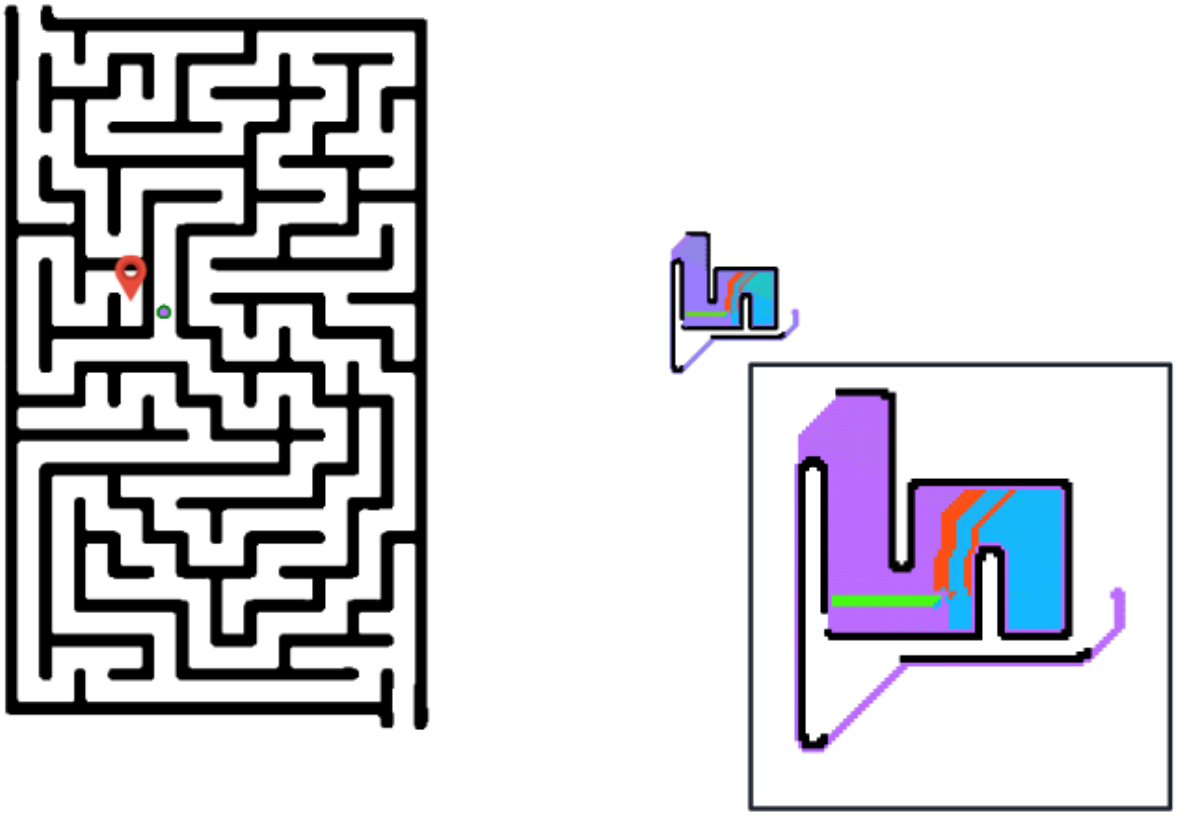

FIGURE 4 – Exploration avec 4 agents sans partage de mémoire.

Le pointeur rouge représente le point de départ des agents et les cercles verts (avec l'agent violet au milieu qui est arrivé en dernier) représentent le but. A droite sont représentées de manière symétrique les traces d'exploration laissées par les agents. On peut voir que nos traces se chevauchent et que les agents ont exploré les mêmes points. On peut maintenant lancer une exploration avec un partage de mémoire pour voir comment vont se comporter nos agents :

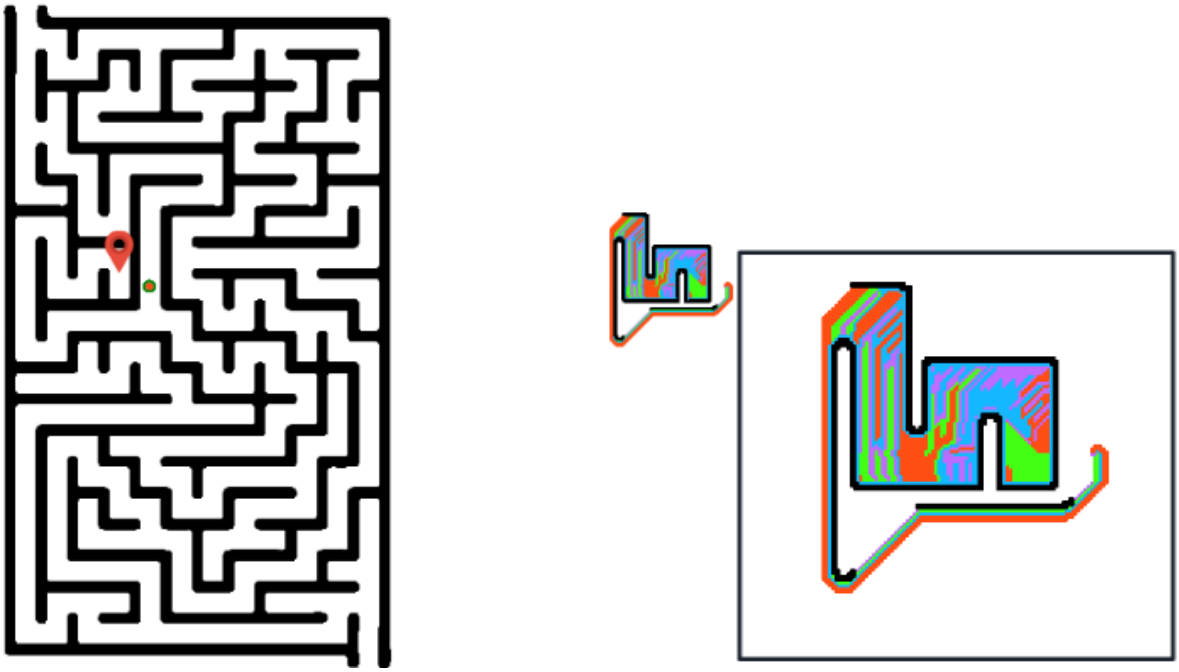

FIGURE 5 – Exploration avec 4 agents avec partage de mémoire.

On peut voir que cette fois-ci les agents ne sont pas repassés sur les traces des autres, on peut voir qu'il y beaucoup plus de couleurs sur cette dernière image. On a également comparé en chiffres nos deux explorations avec le tableau suivant.

	Sans mémoire partagée	Avec mémoire partagée
Agent 1 (Rouge)	3042	1403
Agent 2 (Vert)	3042	1081
Agent 3 (Bleu)	3045	1343
Agent 4 (Violet)	3045	1045
Total	**12174**	**4872**

TABLE 1 – Comparaison des deux stratégies sans/avec mémoire partagée.

On peut remarquer que le nombre total d'action et le nombre d'actions par agents a environ été divisé 2,5. Les résultats que nous observons sont encourageants pour la suite des expérimentations.

4.2 Labyrinthe rectangulaire

Dans l'exemple précèdent, nos agents étaient placés dans une position de départ idéale pour un système à partage de mémoire. En effet, nos agents étaient très proches et dans un espace clos. Nous allons voir ce qui se passe si l'on disperse les agents dans le labyrinthe. Les quatre marker représentent les 4 points de départ de nos agents.

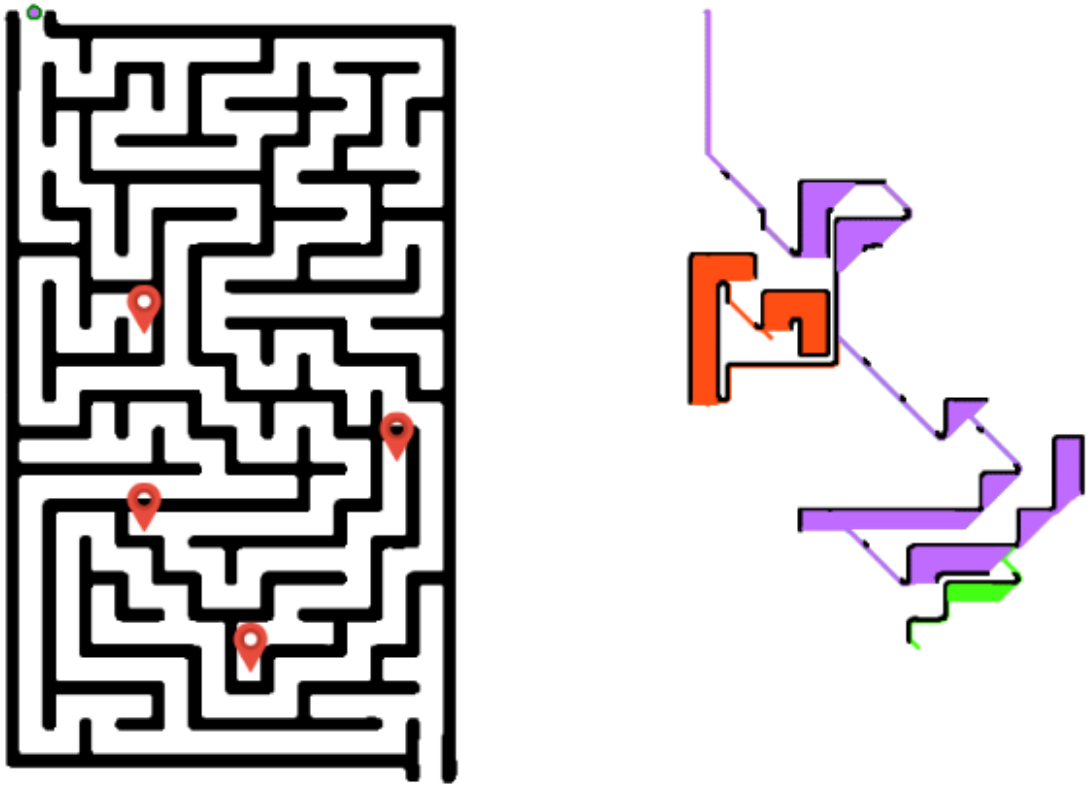

FIGURE 6 – Exploration avec 4 agents sans partage de mémoire.

On peut remarquer que globalement nos agents se comportent comme attendu et qu'ils ont exploré l'environnement chacun dans leur coin. En revanche, l'exploration avec mémoire partagée ne sera pas aussi efficace dans cette situation que dans la précédente. Car, comme nous l'avons vu dans l'exemple ci-dessus, les agents ne retournent pas sur les espaces déjà explorés et dans cette situation certains passages vont être bouchés par des agents qui sont passés avant. Et si le passage bouché mène à la solution, notre agent va partir dans la mauvaise direction comme le montre la figure suivante.

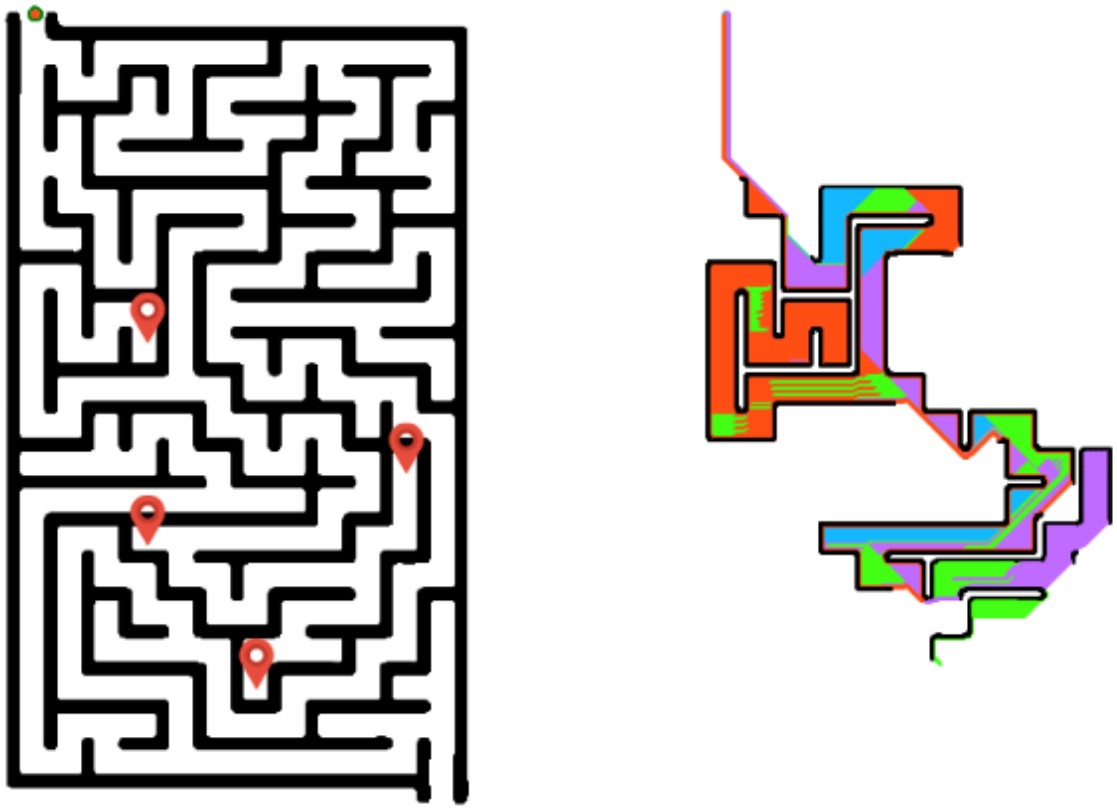

FIGURE 7 – Exploration avec 4 agents avec partage de mémoire.

	Sans mémoire partagé	Avec mémoire partagé
Agent 1 (Rouge)	5402	9411
Agent 2 (Vert)	5570	6222
Agent 3 (Bleu)	3924	3787
Agent 4 (Violet)	6044	8021
Total	**20904**	**27441**

TABLE 2 – Comparaison des deux stratégies sans/avec mémoire partagé.

On voit que, dans cette situation, le partage de mémoire entre nos agents ne sera pas efficace, car les premiers agents vont boucher des chemins qui sont nécessaires à certains agents pour accéder à leur but. Dans ce cas, on a 7000 actions de plus, mais on peut noter que nos agents connaissent mieux l'environnement car ils ont exploré des chemins que les agents sans partage de mémoire n'auraient pas vu.

4.3 Labyrinthe rond

Pour cette partie, on a décidé de changer la forme du labyrinthe pour tester notre stratégie dans d'autres situations. On va procéder aux mêmes tests que précédemment avec cette fois-ci des agents qui partent du même endroit.

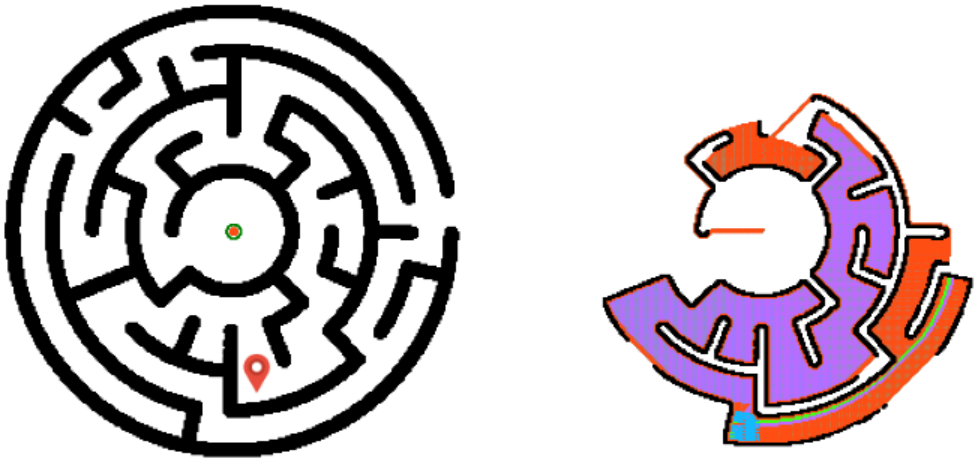

FIGURE 8 – Exploration avec 4 agents sans partage de mémoire.

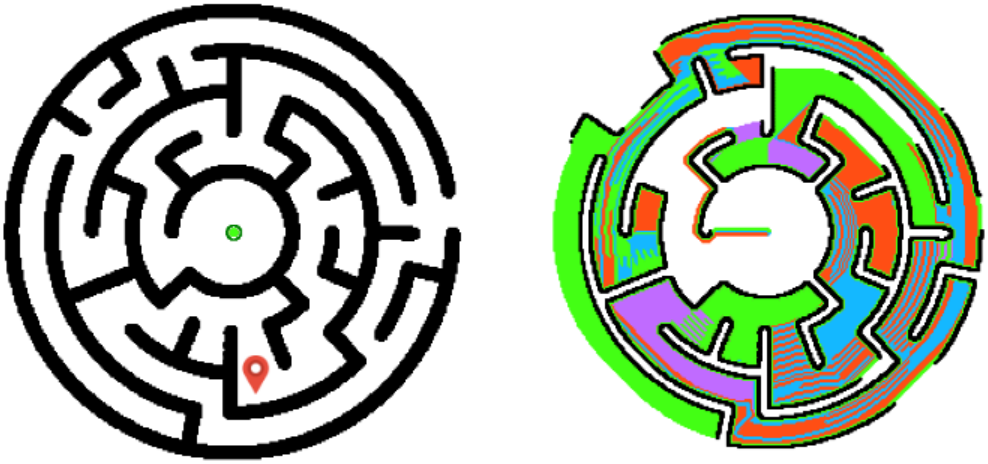

FIGURE 9 – Exploration avec 4 agents avec partage de mémoire.

	Sans mémoire partagé	Avec mémoire partagé
Agent 1 (Rouge)	17347	10887
Agent 2 (Vert)	17343	11060
Agent 3 (Bleu)	17343	8801
Agent 4 (Violet)	17342	3013
Total	**52032**	**33761**

TABLE 3 – Comparaison des deux stratégies sans/avec mémoire partagé.

Cet exemple montre le potentiel de cette stratégie d'exploration, on voit très clairement que les agents qui partagent leur mémoire font moins d'actions pour arriver à leur but et la connaissance totale est plus importante. On peut également noter que nos agents connaissent presque tout le labyrinthe à la fin. Pour parler des chiffres, on gagne 20 000 actions entre les deux stratégies et aucun agent n'a mis plus de 11 100 actions pour atteindre son but. L'agent 4 a mis 3013 actions pour rejoindre le but ; c'est vraiment peu comparé au 17 000 actions exécutées quand il ne partage pas sa mémoire.

5 Conclusion

Pour conclure, notre stratégie peut se révéler être très efficace dans certains cas et dans d'autres non. Pour aller plus loin, on pourrait éventuellement mettre en place un algorithme de pathfinding qui pourrait trouver un chemin à partir de la base de connaissance des agents. Cela pourrait nous permettre de rediriger les autres agents à partir du moment ou un agent à réussi à atteindre son but. Néanmoins, ce projet nous a permis d'apprendre beaucoup de chose sur les systèmes multi-agents.