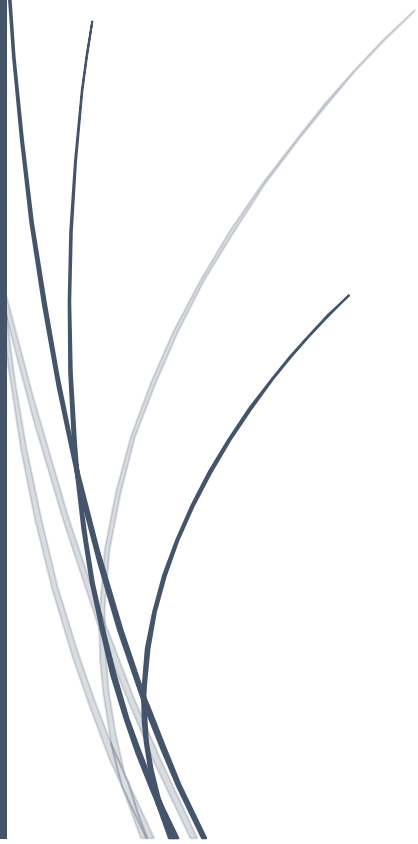
04.12.2022

Rapport du projet

Multi-Agent Path Finding



HERASYMENKO Vladyslav
RUBIO CARJAVAL Andrés Eloy
PALAISEAU, 2022

SOMMAIRE

1. Le contexte

La navigation d'une équipe d'agents dans un environnement partagé est un problème important pour de nombreux domaines d'application existants et émergents. Les exemples de tels domaines incluent : la logistique d'entrepôt, le tri du courrier, la gestion autonome des intersections, la coordination d'essaims de drones etc. Dans chacun de ces contextes, on fait face à un problème combinatoire NP-difficile connu sous le nom de Multi-Agent Path Finding (MAPF).

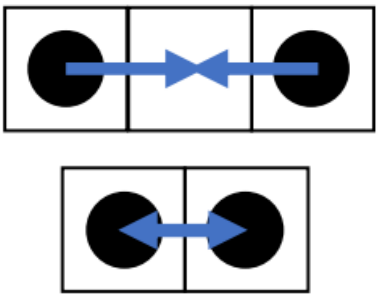
Les études sur ce sujet apparaissent souvent dans la littérature de l'intelligence artificielle et dans les actes des conférences phares, comme l'AAAI. Ces études concernent également les chercheurs de domaines adjacents, tels que la robotique et l'optimisation discrète.

Dans le cadre de ce projet, nous nous sommes appuyés sur le tutoriel « Tutorial on Recent Advances in Multi-Agent Path Finding », publié en AAMAS et qui propose un aperçu des techniques courantes de problèmes de MAPF.

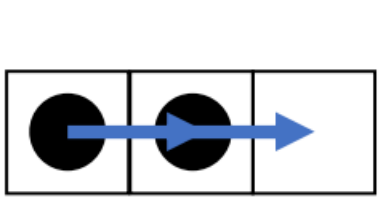
2. La problématique

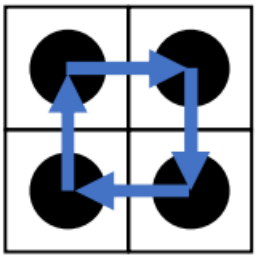
Commençons par poser le problème de MAPF. Sommairement, l'idée consiste à faire . Toutefois, certaines contraintes sont appliquées.

Premièrement, les collisions où les deux agents se retrouvent sur une même case de la grille sont interdites. Il y a deux types de tels collisions, comme illustré ci-dessous :



Néanmoins, le mouvement « en chaîne » est autorisé.





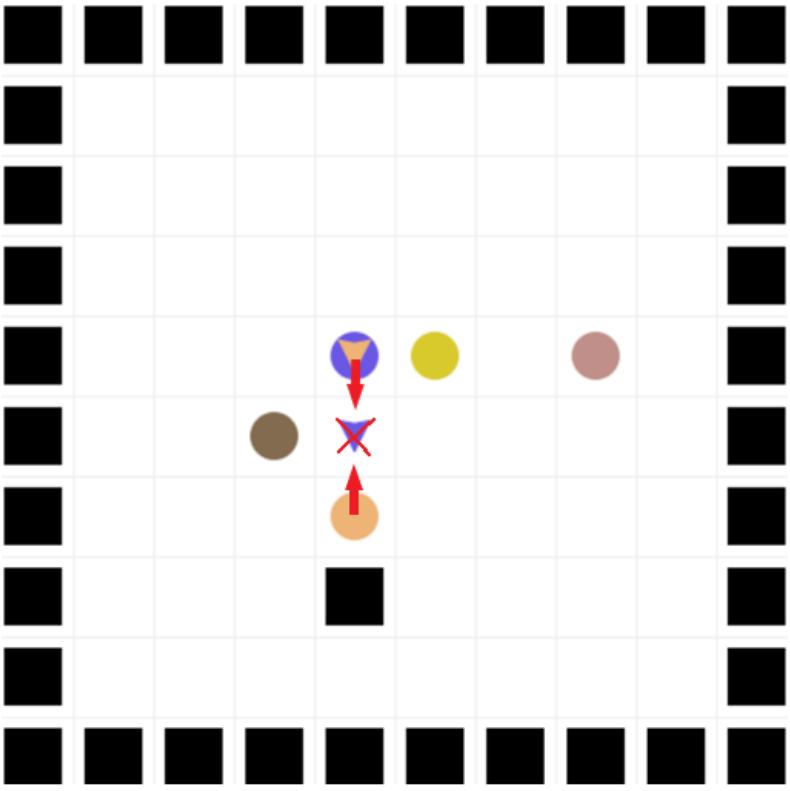
En outre, l'acheminement doit être effectuée de sorte à minimiser le coût (dans le cas le plus simple, le coût est la somme des pas effectués par tous les agents)

3. Les solutions testées

Le tutoriel propose un grand nombre des solutions possibles au problème décrit dans le chapitre précédent. Nous nous sommes intéressées aux trois modifications de l'algorithme A*, notamment : A* classique, A* coopératif et A* avec le Stochastic Local Search.

Tout d'abord, on a commencé avec l'approche la plus facile qui consiste à appliquer niaisement l'algorithme classique de A* (dans notre cas, nous utilisons l'algorithme de Dijkstra qui est un cas particulier de A*) à chaque agent un par un pour trouver un chemin vers leurs cibles, sans prendre en compte les actions des autres agents.

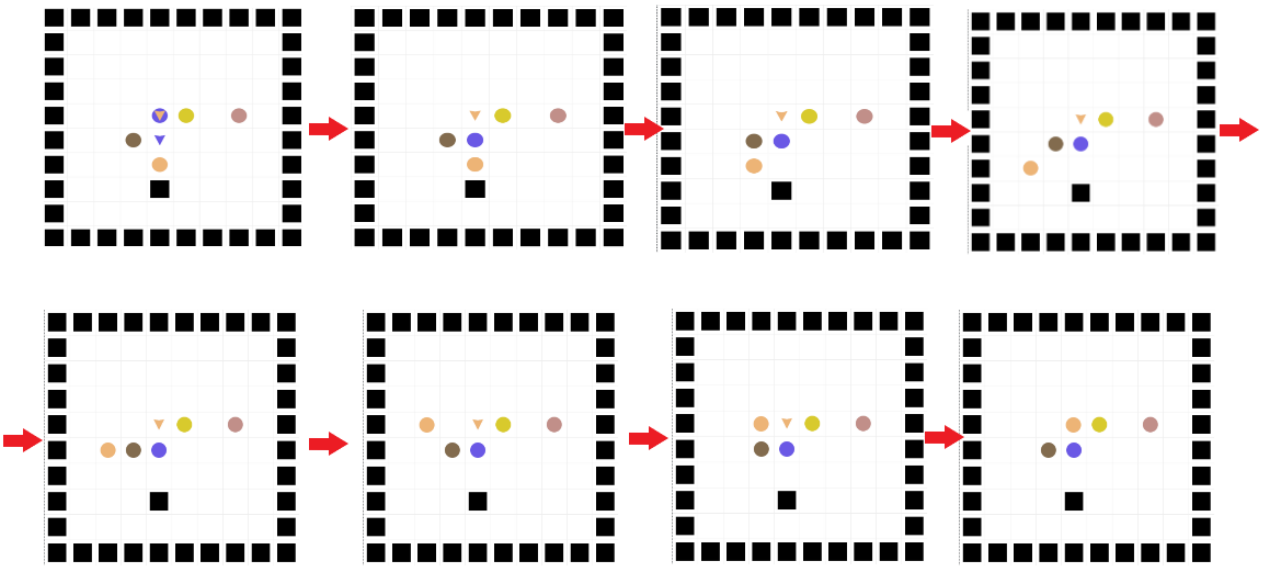
Cela marcherait assez bien pour des problèmes de petites dimensions, mais plus le nombre des agents est important, plus la probabilité de conflits (est surtout des dreadlocks) est élevée. Par exemple, dans le cas suivant l'agent violet veut descendre en bas et l'agent orange doit aller en haut, passant par la cible de l'agent violet.



Cette situation engendre un deadlock, car les deux agents doivent passer par une même case, bien que l'algorithme de Dijkstra ne soit pas capable de circonvenir ce genre de conflits.

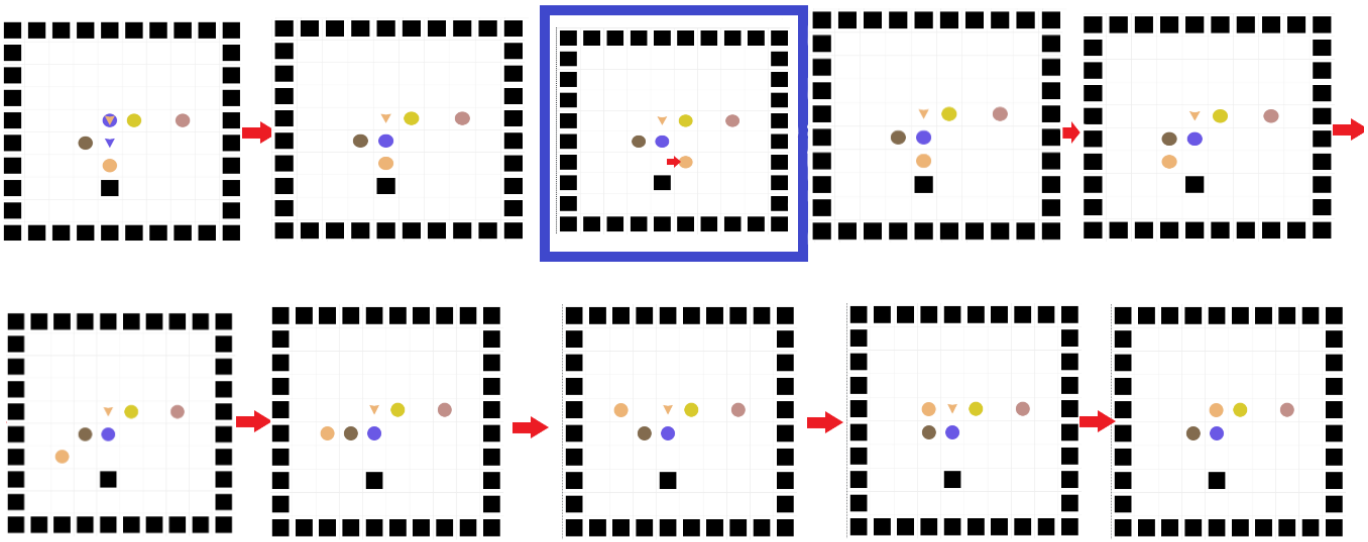
Pour adresser ce type de problèmes, il est proposé dans le tutoriel de modifier légèrement l'algorithme classique. Plus précisément, les agents ont désormais la

connaissance parfaite des positions et des plans des autres agents, donc à chaque tour de simulation, les chemins vers les cibles peuvent être recalculés en prenant en compte les positions courantes et futures (sur le pas suivant) de chaque agent comme si c'était des obstacles. Ainsi, on peut obtenir une solution quasiment optimale sans trop modifier l'algorithme en soi (d'ailleurs cette modification s'appelle A* coopératif). Néanmoins, en pratique il est très difficile d'assurer le partage parfait des informations entre tous les agents. Il est, de plus, nécessaire de recalculer le chemin de chaque agent à chaque tour, ce qui nécessite un nombre très important de calculs. La résolution du conflit précédent est illustrée sur l'image ci-dessous.

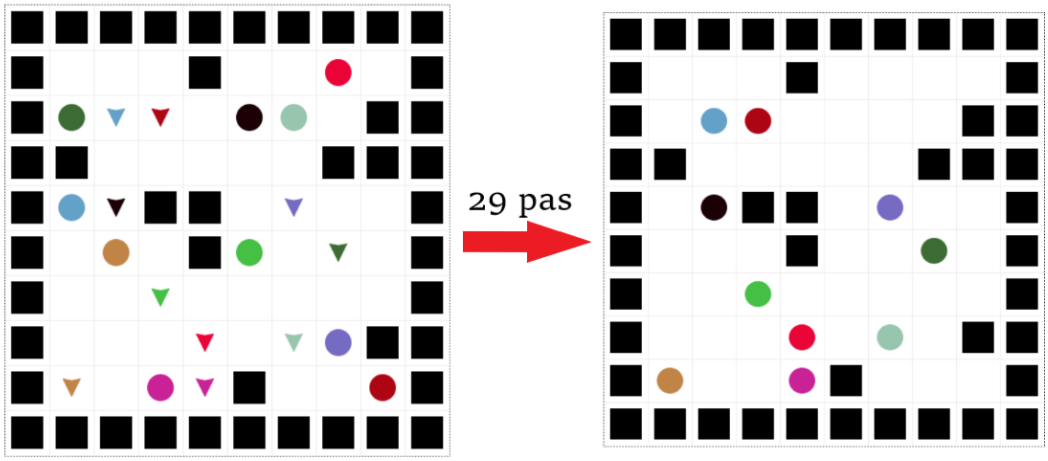


Enfin, une autre approche pour résoudre le problème de deadlocks est le Stochastic Local Search. Dans le cadre de ce projet, on utilise une version simplifiée de cette méthode qui consiste essentiellement à tout simplement faire un pas aléatoire lorsqu'on fait face à une situation de blocage (quand deux agents doivent occuper la même case). De cette manière, il ne faut recalculer le chemin des agents qu'après qu'un agent fait un pas aléatoire pour éviter la collision ce qui réduit significativement le temps de calcul. L'inconvénient de cette approche, toutefois, est le fait que l'optimalité n'est plus garantie.

Illustrons comment le problème de deadlock va être résolue à l'aide du Stochastic Local Search.



On peut également appliquer cette méthode à des problèmes de plus grandes dimensions. En général, cela donne des très bons résultats, comme sur l'image suivante :



En quelque sorte, on peut dire que l'algorithme de A* coopératif, appliqué dans le contexte de problème de MAPF, représente intrinsèquement une architecture des agents coopératifs, car ils communiquent entre eux et doivent modifier leur comportement (le chemin) en fonction des actions des autres agents. Par contre, le Stochastic Local Search ressemble plutôt l'approche des agents réactifs, car, dans ce cas, les agents n'ont aucune représentation interne de l'environnement et n'interagissent avec lui qu'à travers des « capteurs » qui ne prennent en compte que leurs alentours immédiats.

4. Les conclusions

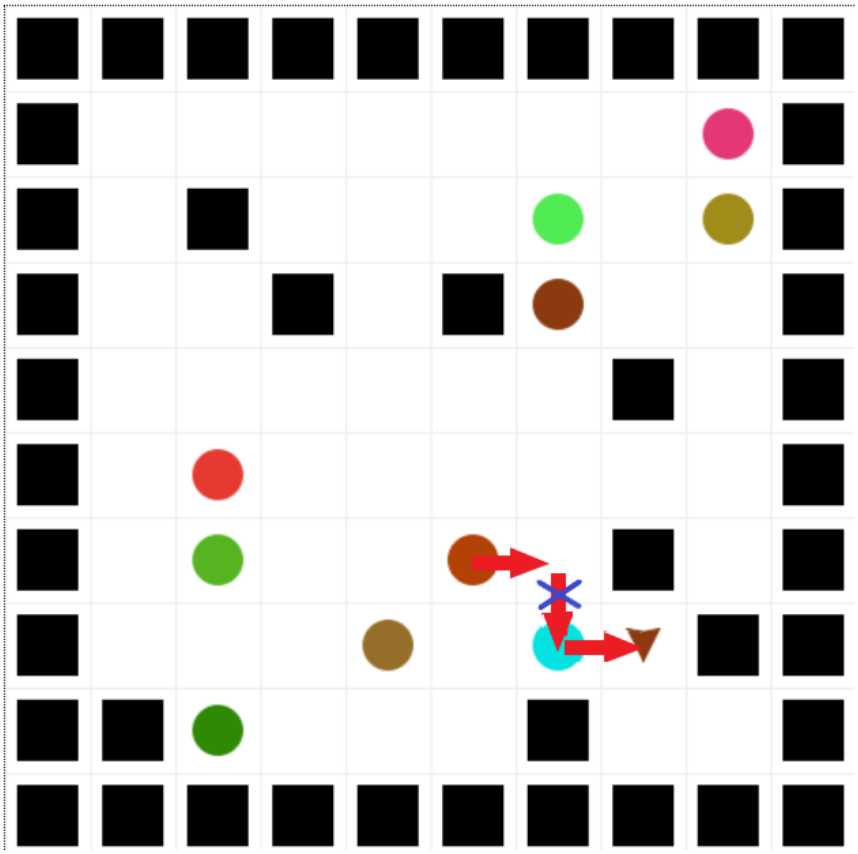
En résumé, il n'est pas possible de conclure quel algorithme serait le meilleur dans le cas général, car les trois méthodes ont ses propres avantages et inconvénients.

Le A* classique donne la solution optimale et ne nécessite pas beaucoup de calculs, mais ne marche sans blocages que pour des problèmes de petites dimensions.

Le A* coopératif, d'un autre côté est capable de résoudre des conflits et donne une solution quasiment optimale. Toutefois, il nécessite une connaissance parfaite de toutes les positions de tous les agents et, de plus, il faut effectuer un nombre très important de calculs à chaque tour de simulation.

Finalement, le Stochastic Local Search se trouve un peu au milieu entre le A* classique et A* coopératif. Il est plus simple en termes de calculs que ce dernier, ne nécessite pas la connaissance des positions des autres agents, mais est moins robuste pour la résolution des conflits.

Enfin, il faudrait également mentionner qu'il y a un type des conflits qui ne peuvent pas être résolus par les méthodes qu'on a utilisées : le blocage des bottlenecks. Ce type des verrous survient, par exemple, lorsqu'un agent occupe indéfiniment la case d'un passage étroit (e.g. quand il atteint sa cible), tandis qu'un autre agent doit passer par cette case pour atteindre sa cible à lui, comme illustré sur l'image ci-dessous :



En ce qui concerne les aspects des systèmes multi-agents qu'on a dûs apprendre, sans doute a-t-on dû, avant tout, comprendre la tâche de MAPF et les algorithmes de Path Finding qui y sont appliqués. Outre cela, il nous fallait également se familiariser avec un certain nombre des fonctions et des modules de Mesa, en Python, qui n'étaient pas présentés dans le cadre du cours, vu qu'ils sont assez spécifiques.

5. Les apports de chaque membre de l’équipe

Vladyslav - Les codes
Andrés - La bibliographie, les code-reviews et le refactoring du code