Cooperative Pathfinding

Angelo Ortiz
Sorbonne Université

Introduction

De nos jours, c'est un fait que les robots se sont répandus dans les industries et ils ont ainsi remplacé, parfois complètement, l'humain dans certaines tâches. Par exemple, ils sont de plus en plus utilisés dans les actes chirurgicaux, comme la laparoscopie, du fait de leur précision. Pour d'autres tâches, comme le déplacement de marchandises ou des objets lourds dans les usines, il est aussi plus intéressant de les attribuer aux robots.

Mettons-nous dans cette dernière situation. Nous avons une entreprise qui compte plusieurs robots chargés de déplacer des marchandises à l'intérieur d'une usine. Il est évident que l'objectif est d'effectuer des déplacements de manière efficiente, i.e. des trajets plus courts. Supposons de plus que les objets à ramasser sont répartis entre l'ensemble de robots de sorte qu'il n'y ait qu'un seul robot ciblant un objet.

La première réponse naïve au problème des déplacements est de calculer, pour chaque entité, le plus court chemin à son but compte tenu des obstacles et de les faire emprunter ces chemins. En effet, plusieurs algorithmes de très bonne complexité temporelle sont connus à cet effet. Cependant, nous nous apercevons que cette approche n'est pas efficace car elle néglige les potentielles collisions qui peuvent survenir en cours de route.

Cela nous amène à concevoir une autre stratégie tenant compte des possibles collisions pour résoudre ce problème. Il s'agit ainsi du problème de la recherche coopérative (en anglais *Cooperative Pathfinding*).

Dans le cadre de l'UE 3I025 - Introduction à l'intelligence artificielle et la recherche opérationnelle, le sujet du premier mini-projet proposé prend la forme du problème du *Cooperative Pathfinding* et il consiste à résoudre ce problème au travers des différentes stratégies appelées opportuniste, coopérative de base et coopérative avancée.

Dans la suite de ce document, je vais expliquer en détail les spécifications du sujet, mes choix d'implémentation et commenter mes résultats obtenus.

Présentation

Dans le cadre de ce projet, le problème du *Cooperative Pathfinding* prend la forme d'une grille rectangulaire à deux dimensions. Elle comporte trois types d'objets : les joueurs, les fioles et les murs. Un joueur peut se déplacer sur la carte, prendre une fiole s'il y en a une sur sa case, mais ne peut pas traverser les murs ni les autres joueurs, ce que l'on appelera une *collision*.

Trois types de collisions ont été définis :

(I) la *concurrence*, où deux agents veulent tous les deux se diriger vers la même position à l'instant suivant ;

(II) le *croisement*, où deux agents se traversent l'un l'autre en un instant de temps ;

(III) la *poursuite*, où un agent se dirige vers un emplacement déjà occupé par un autre agent qui avance dans son chemin en ne traversant le premier agent.

Les deux premières collisions sont dites *frontales*, puisqu'elles n'arrivent que lorsque les deux agents se trouvent face à face.

Par ailleurs, le problème a été considéré comme étant à information complète. Autrement dit, les agents connaissent l'intégralité de la carte, ce qui correspond à l'emplacement des obstacles, et la position de leurs pairs et leurs trésors associés. Nonobstant, un joueur ne peut récupérer que la fiole qui lui a été associé au départ.

Le but est alors de trouver les plus courts chemins des agents à leur fiole respective tout en évitant les collisions entre les agents. À cet effet, trois stratégies ont été proposées : la stratégie opportuniste, la coopération de base et la coopération avancée.

Il est important de remarquer que l'ordre de passage des agents est défini et fixé lors de l'instanciation du problème : il correspond à l'ordre de création des agents.

Stratégies

La base de toutes mes stratégies de résolution du problème du *Cooperative Pathfinding* est la recherche du plus court chemin dans une carte avec obstacles. Pour ce faire, j'ai utilisé l'algorithme A*. Puis, j'ai implémenté trois façons de gérer les potentielles collisions sur la carte : elles correspondent aux stratégies détaillées dans les sections qui suivent.

Pour la première et la troisième approche, j'ai pris en compte uniquement les collisions frontales pour faciliter le recalcul de chemin. Quant à la deuxième, les trois types de collisions ont été considérés du fait de sa conception stricte.

Opportunisme

La première stratégie, dite *opportuniste*, consiste à démarrer le chemin de tous les agents en même temps et de gérer les collisions lorsqu'elles sont imminentes. Pour ce faire, j'ai utilisé la méthode connue comme *path splicing*.

Lorsqu'un agent détecte que la suite de son chemin l'amène vers une collision à l'instant suivant, il décide de recalculer une partie de son chemin. Pour ce faire, il utilise de nouveau l'algorithme A* jusqu'au bout de la portion de chemin à remplacer en tenant compte des positions bloquées par la collision. Puis, il joint les deux morceaux de chemin et suit ce nouveau chemin. Ce comportement implique que la gestion des collisions est laissée à la charge du premier agent (dans l'ordre de passage) concerné.

J'ai trouvé deux points importants à remarquer dans cette méthode. D'une part, la longueur de la portion de chemin à recalculer joue un rôle crucial. En effet, si l'agent est amené à calculer des longs morceaux de chemin et que les collisions sont fréquentes, il passera la majorité de son temps à faire des calculs inutiles, puisqu'il ne suivra qu'une infime partie des chemins calculés. C'est pourquoi, tout au long de mes tests, j'ai fixé la *longueur de coupure* de l'ordre de 5.

D'une autre part, il se peut que le chemin restant pour atteindre la fiole respective ne soit pas plus long que la longueur de coupure fixée. Dans ce cas, l'agent supprime son chemin restant et fait un pas aléatoire valide dans l'une des positions adjacentes, y compris sa position courante. Il s'avère que cette *randomisation* allège le temps de calcul et simplifie la suite, étant donné qu'elle repousse la recherche du chemin restant à l'instant suivant où les agents concernés se trouvent dans d'autres positions. De plus, le pas aléatoire est très utile lorsque la position de collision se trouve dans une zone concentrée d'agents, et surtout lorsqu'il est nul, i.e. l'agent reste immobile, dans quel cas le coût associé est 0.

Coopération naïve

planification

Coopération avancée

Finalement, la troisième approche, dite *avancée*, est une implémentation de l'algorithme *Windowed Hierarchical Cooperative A** proposé par David Silver.

Dans cette approche, la carte des joueurs devient un espace tridimensionnel où le temps a été rajouté. De plus, l'heuristique de la distance de Manhattan est remplacée par la distance réelle entre toute position et l'objectif. Pour ce faire, j'ai utilisé une version modifiée de l'algorithme A* spatial où la position initiale est celle de la fiole cherchée et dont l'objectif peut être modifié en cours d'exécution. Ceci a réduit le temps mis par les itérations de A* tridimensionnel car il stocke en mémoire la recherche de A* spatial et n'a donc besoin de le relancer que lorsqu'un nœud pas encore visité est atteint.

En ce qui concerne la gestion des collisions, une table de réservation des positions dans le temps est mise en place. Lorsqu'un joueur lance sa recherche de A* spatio-temporel, il considère les positions réservées par ses pairs dans la table et réserve lui-même les cases correspondant à son chemin optimal. De la même manière que pour la stratégie opportuniste, un agent n'encourt aucun coût s'il reste immobile.

Comme la recherche de A* inversé est stockée en mémoire, il est raisonnable de l'effectuer par morceaux. De plus, tous les agents anticipent les collisions, et donc certaines mesures lointaines peuvent ne pas être nécessaires lorsque arrivés à ce point, ce qui les rend inutiles. De ces faits, la recherche d'un agent est effectuée par *fenêtre fixe* : elle n'est pas exécutée jusqu'à l'atteignement du but, mais pour un nombre de pas fixés appelé *fenêtre*, et ce, même si l'agent arrive à la position de sa fiole avec moins de pas.

Pour que ces recherches soient efficientes, mais que la présence de l'agent dans la table de réservation reste considérable, j'ai fixé la *fréquence de recherche* à la moitié de ladite fenêtre. Cependant, cette méthode reste encore sous-optimale du fait que les derniers agents à passer ne disposent que d'un nombre limité de choix lors de leurs recherches. Pour y remédier, les recherches sont intercalées.

Les agents sont classés dans divers groupes comportant à peu près le même nombre de joueurs de telle sorte qu'à chaque instant, un seul groupe fait sa recherche. Cette méthode circulaire réduit le désavantage d'un agent à son groupe, puisque cet ensemble sera à un moment le premier à réserver des positions. En outre, cette approche améliore la coopération des agents parce que l'agent qui a déjà réussi son but se verra obligé de se décaler dans son avenir proche lorsque l'un des autres aura réservé au préalable la position sur laquelle il se trouve actuellement.

J'ai constaté que la valeur de la fréquence des recherches a un gros impact sur la performance de cette stratégie. En effet, pour des valeurs supérieures à 6 le temps CPU et le nombre de pas nécessaires, tous les agents confondus, deviennent beaucoup plus grands et le rendu graphique est moins fluide, ce qui correspond à l'augmentation du nombre d'itérations de A* spatio-temporel.

Conclusion

Première stratégie :

Cette stratégie m'a permis de résoudre la plus part des collisions, mais le nombre de pas en moyenne pour rammasser toutes les fioles est assez elevé.