

LICENCE INFORMATIQUE

RAPPORT DE MINI-PROJET D'IARO

Cooperative Path-finding

3i025, Licence d'Informatique L3

REALISER PAR:

DJEDDAL Hanane.

TOUZARI Leticia.

2018/2019

INTRODUCTION

Le problème de path-finding peut se ramener à un problème de recherche du meilleur chemin entre deux nœuds dans un graphe. Il existe un ensemble d'algorithmes classiques pour résoudre ce type de problème.

Toutefois, le path-finding devient un problème complexe lorsque l'on cherche à prendre en compte diverses contraintes additionnelles (exécution en temps réel, présence d'incertitudes, contrainte de ressources, environnement évolutif, ...etc.).

Dans le cadre de ce mini projet, on considère en compétition plusieurs personnages qui doivent chacun atteindre un objectif (une fiole donnée) tout en évitant des collisions éventuelles entre eux.

L'objectif étant donc de minimiser le temps total nécessaire à la récupération de toutes les fioles, trois approches de path-finding seront étudiées dans ce contexte.

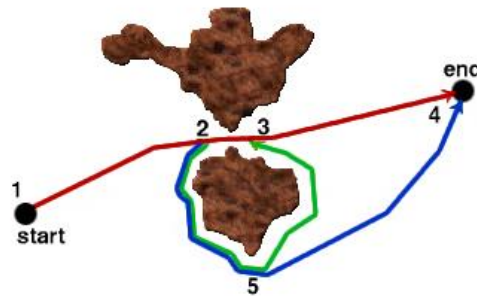
1. Stratégie opportuniste de portionnement de chemins ("path slicing") :

Principe :

Cette méthode permet de retrouver un chemin optimale ($C_1 \dots C_i \dots C_n$) pour le joueur en utilisant l'algorithme A^* , mais cela n'empêche qu'au moment où le joueur atteint une case particulière, un nouvel obstacle soit peut-être à cette case « C_i » (particulièrement une collision avec un autre joueur). Le pathfinder fait donc en sorte de contourner les obstacles tout en suivant son chemin initial autrement dit au-lieu de déplacer le joueur vers la case C_i qu'un nouvel obstacle occupe, désormais un nouveau chemin est recalculé à partir de la case C_{i-1} vers C_{i+1} (avec le même algorithme, ici A^*) qui sera par la suite concaténé au chemin initial à partir de la case C_{i-1} ($C_{i-1} \ x_1 \ x_2 \dots x_k \ C_{i+1} \dots C_n$).

Performance:

Lorsqu'un chemin doit être recalculé, cela signifie qu'une collision (nouvel obstacle) est détecté à la prochaine case. En contournant l'obstacle il peut arriver que le nouveau chemin soit long et pas très bon. La figure ci-jointe illustre une telle situation :



De plus le nouveau chemin ne garantit pas qu'il n'y aura pas de nouvelles collisions sur le chemin du contour ce qui voudrait dire que le joueur se verra recalculer un nouveau chemin en plein contour du précédent et donc une possibilité de s'éloigner d'autant plus du chemin initial pour au final retourner vers ce chemin.

2. Stratégie coopérative de base:

Principe :

Dans cette approche il s'agit d'identifier des chemins qui ne partagent aucune case (aucune collision possible) pour former des groupes de joueurs. Pour chaque groupe, les chemins peuvent être exécutés en parallèle par les joueurs en étant certain de ne pas entrer en collision. Le défi de cette stratégie est de trouver la répartition optimale des joueurs. Dans notre cas, un chemin optimal est calculé pour chaque joueur (utilisant l'algorithme A*) puis un parcours des groupes déjà existant est effectué pour retrouver le groupe dont les chemins ne se croisent pas avec le chemin du joueur, si plusieurs groupes sont compatibles, ce joueur est rajouté au groupe ayant le plus grand temps d'exécution (le plus long chemin) pour ne pas augmenter le temps exécution total du groupe, sinon dans le cas où aucun groupe n'est compatible un nouveau groupe est créé lui affectant ce nouveau joueur.

Performance:

Le principal inconvénient de cette méthode est de considérer que deux chemins entrent en collision sous le seul prétexte qu'ils partagent des cases communes, or le facteur temps est un critère primordial pour affirmer une collision, ce qui donc oblige des joueurs à faire des parcours en séquentiel lorsque des parcours en parallèle sont possibles.

3. Stratégie coopérative avancée :

Principe :

Dans cette approche on s'inspire des deux approches précédentes : On détecte les chemins qui peuvent se croiser en prenant en considération non seulement les cases

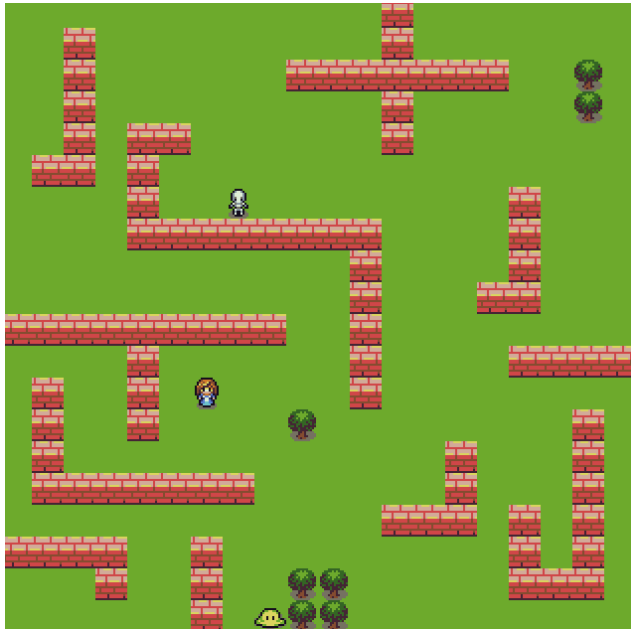
communes mais aussi une troisième dimension qui est le temps ; et on recalcule les chemins en considérant les autres comme des obstacles, mais contrairement à la première stratégie, ce calcul est fait avant de commencer le parcours, sous forme d'une planification ; ce qui permet de prévoir les futures collisions résultantes d'un deuxième calcul.

Dans notre implémentation, on commence par transformer notre espace, initialement deux-dimensionnel, à un espace trois-dimensionnel avec le temps comme troisième dimension. Alors une case (i, j) n'est plus définie seulement par sa ligne i et sa colonne j , mais aussi par une troisième valeur t représentant le temps. Cette dernière valeur vaut un caractère spéciale (-1 pour nous) pour les obstacles permanents. Ensuite, on utilise une table de réservation, initialement vide, qui est mise à jour au fur et à mesure des calculs des chemins de différents agents ; et donc lors du calcul de chemin d'un agent a_i , les chemins c_k ($k = 1 \dots i-1$), stockés dans la table de réservation, sont considérés comme des obstacles dans notre espace 3D.

Performance:

Pour des exemples simples, cette stratégie donne des résultats similaires à ceux de la première. La différence se voit quand la résolution d'une collision engendre une autre qui elle-même engendre une autre jusqu'à niveau indéfini. Cette situation résulte en une boucle infinie avec la première stratégie. Or la troisième stratégie, en planifiant à l'avance, prend toutes les cases réservées en considération et, par la suite, trouve un chemin possible ou bien, si aucun chemin est trouvé, décide de faire une *Pause*.

Tests:

| Carte | Temps de parcours pour chaque stratégie |
|-------------------------------------------------------------------------------------|----------------------------------------------------------|
|  | Stratégie 1 : 80 Stratégie 2 : 56 Stratégie 3 : 52 |



Stratégie 1 : 43
Stratégie 2 : 43
Stratégie 3 : 43



Stratégie 1 : 39
Stratégie 2 : 59
Stratégie 3 : 31



Stratégie 1 : 208
Stratégie 2 : 197
Stratégie 3 : 173

Comparaison entre les stratégies:

La première stratégie, étant la plus spontanée, offre un temps d'exécution acceptable (dans les situations simple) qui peut croître en fonction du nombre des collisions possibles. De plus, les calculs étant faits au fur et à mesure de l'avancement, la transition d'un agent d'une situation à une autre peut prendre un plus du temps. Finalement, elle peut engendrer des boucles infinies.

La deuxième stratégie permet de résoudre le problème de terminaison et offrir un avancement fluide des agents, les calculs étant faits avant le début des parcours. Par contre, elle n'exploite pas le parallélisme à son max, et nécessite un calcul ainsi qu'un espace mémoire de plus.

Finalement, la troisième stratégie, en ajoutant une notion supplémentaire : le temps, permet de résoudre pas mal de problèmes. Mais cette dimension de plus ainsi que la table de réservation occupent un espace mémoire qui peut devenir vite considérable. Le temps de calcul devient aussi un facteur, ce qui a donné naissance à des améliorations qui limite le niveau de la recherche.

CONCLUSION

En se basant sur l'algorithme A^* , qui donne des résultats parfaits pour un seul agent, on a pu arriver à des solutions plus complexes pour le problème de pathfinding quand plusieurs agents sont présents. L'approche opportuniste, où chaque agent prend des décisions locales, donnait un temps d'exécution acceptable mais seulement pour des configurations peu complexes. En passant à des approches coopératives, où les agents peuvent communiquer et donc une coordination est possible, on a pu améliorer les performances et résoudre le problème de la terminaison posé par la première approche. Finalement, l'introduction d'une troisième dimension a ramené le problème à un autre niveau où de nouvelles améliorations sont introduites.