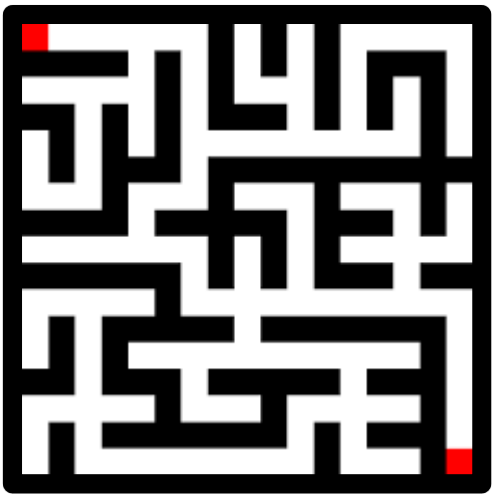
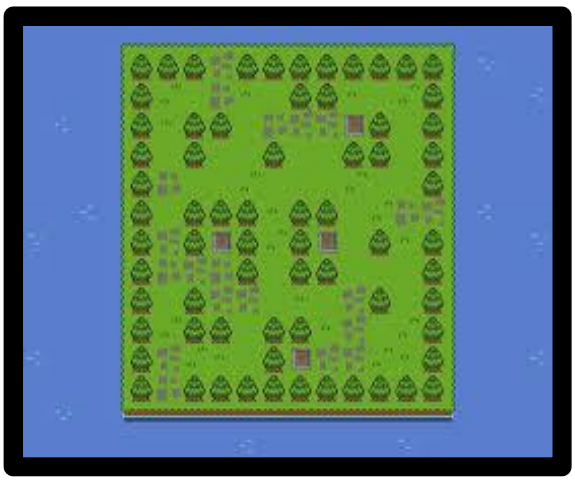
Flavien AUBARD-POILLOT et Thomas ORLUC Groupe : coop-pathfinding-ab_or

Mini-Rapport de projet 3i025

Explication de code sur le sujet :

''Le pathfinding lors d'une situation de coopération ''





Année 2018-2019 L3 S6

1)L’algorithme Astar :

L’ago astar que nous avons implémenté est assez simple et se divise en plusieurs étapes.

Tout d’abord parlons de ses paramètres :

- initState qui est tout simplement la position de départ de l’algorithme de recherche, elle permet lors de la première itération d’étendre la liste des coups à explorer.

- goalState qui est la valeur de la case que l’algorithme va chercher, c’est l’objectif d’arrivée de cet algorithme.

- wallState est le dernier paramètre de la fonction et est une liste qui stock la position des cases que l’algorithme ne doit pas se préoccuper c’est-à-dire que l’algorithme ne devra pas passer par cette case.

Un coup est composé d’un couple de coordonnées, d’un entier qui stock le combien – ieme coup il représente dans le chemin, d’un score qui prend en compte la distance de Manhattan entre la position et le but et enfin d’une référence au coup d’avant (au moment où il a été entendu).

Maintenant l’algo en lui-même :

On initialise une liste des coups exploré ‘’coupexplore ’’ et on y ajoute le coup initial dedans avec en référence au coup d’avant égale à NULL.

Cette liste permettra plus tard de vérifier si le coup a déjà été joué plus tot. On prend ensuite la liste des coups aux alentours de la case ou l’algorithme est. C’est-à-dire pour notre cas (0,1) (0,-1), (1 0), (–1 0) car on ne peut pas avancer en diagonal. On vérifie si les nouvelles coordonnées initiées à newlig newcol sont ajoutable a la liste des coups avec les tests du if (coordonnées pas déjà explorées pas dans le wallstates et comprise entre 0 et 20).

Et là, deux cas se présente à nous, soit la cordonnée est le goalState et si c’est le cas on crée une liste de coups et on va remonter jusqu’à la position de départ (jusqu’à arriver à un NULL).

Une fois la liste crée on la reverse (sans utiliser reverse car la fonction reverse en python reverse aussi les coordonnées de la liste. On utilise donc une simple boucle. On **return** alors la liste.

Le second cas et le plus souvent présent est celui ou on est sur une autre case que celle d’arrivée. Dans ce cas-là on initialise le coup avec les valeurs adaptées, la pose le numéro de coup qui est égale au numéro du coup d’avant +1, le score de la fonction heuristique et ‘’père’’ qi on peut l’appeler comme ça qui permet de stocker le chemin de retour pour le cas du dessus. Une fois tous les nouveaux coups initialisés on les ajoute à la frontière qui est la liste des coups à visiter. Le nom de la liste est réserve. (Cet ajout dans la réserve se passe ente la ligne 86 et 91). Enfin on récupère le coup dont le score est le plus bas afin de refaire un tour de boucle avec la nouvelle case visitée à étendre.

2)Les Stratégies de coopération en Multijoueur :

Le main :

Le main est très simple c'est un while true qui permet d'enchainer les recherches.

De base le main initie des positions d'objet et des joueurs qui sont lié a un objet.Il initie le chemin des joueurs avec le Astar et démarre le chemin sur la map. A la base il ne gérait pas les collisions et les joueurs se traversaient, pour cela on a juste ajouté une condition qui permet de détecter cette collision. Et de le print sur la console. De plus lorsque le joueur trouvait la fiole il enchaînait avec une autre fiole. Nous avons corrigé ça avec un simple continue ligne 294/297.

La Détection de collision :

Cette détection se passe entre les lignes 306 et 310, en effet dans la liste des joueurs si une personne est à la position ou le joueur dois aller, alors l'algo le détecte print la collision et entre quels joueur elle a lieu, une fois que la détection était bien implémentée nous avons pu commencer à penser à la gestion de ces collisions.

La Stratégie Opportuniste :

La stratégie opportuniste est assez facile à implémenté.

Pour ce faire nous avons utilisé plusieurs paramètre, une pose init qui est la pose d'avant la collision, le goal state qui ne change pas du astar initial, i le combien-ieme coups ou la collision est apparue, la case ou la collision a lieu ''coup''.

La fonction consiste à faire un nouvel a star avec la nouvelle position en ajoutant le coup de collision au niveau des wallStates. Pour cela on fait une copie de wallState et on add le coup a la nouvelle liste.

On fait une liste res que l'on va retourner à la fin, on lui ajoute (0.0) jusqu'au coup i (on fait du remplissage vu que une fois de retour dans le main on va continuer notre chemin, et on ajoute a res le chemin du nouveau astar.

Cet algorithme marche assez bien cependant nous pouvons tomber sur certain cas de ''danse '' nous en parlerons dans la conclusion.

Nous voulions avec le Boolean et le adv présent dans la liste des paramètre, faire une version plus rentable notamment en gèrent le cas où l'adversaire qui nous bloc a bougé, cependant le temps nous as manqué et l'amélioration apporté est peu concluante. Il s'agissait de vérifier si l'adversaire avait débloqué la case de collision et de refaire le astar pour retourner faire le chemin le plus cours au lieu de potentiellement contourner.

La Stratégie coopérative de base :

Cette stratégie ne s'applique pas de la même manière que l'opportuniste, en effet l'opportuniste était appliqué lorsqu'une collision avait lieu, or celui la remplace la boucle d'initialisation des astar, on appelle donc collisioncoop qui fait une boucle qui passe par chaque joueurs, avec pour le premier joueur un wallState qui est celui initial du main. Si le astar rend un chemin non vide, on ajoute à la copie de wallState les case parcouru par le joueur 1

On passe au joueur 2 qui va faire le Astar avec le nouveau wallState si le chemin est bien non vide c'est qu'il a trouvé une voie alternative pour aller de la case A à B si le rendu de la fonction est vide, on fait un astar classique. Et ainsi de suite et on ajoute les case supplémentaire a wallS la copie.

Cette algo est vraiment simple de base mais très compliqué à optimiser. Car dans le cas d'une map avec beaucoup de mur ou une petite carte beaucoup de fois elle va revenir à faire un simple Astar pour tout le monde. Elle évite donc beaucoup de collision mais nous l'utilisons en complément de la détection de collision et en complément de collisionOp

Une façon de la rendre optimisée serait de voir tous les cas possible d'ordre de passage en formant tous les couples possible avec le nombre de joueurs. Pour 3 il y a 6 cas mais si on augment nbjoueurs la complexité augmente énormément. Nous avons donc jugé plus intéressant de passer à l'algo suivant qui est le plus efficace pour nous.

La Stratégie coopérative avancée :

La dernière stratégie que nous avons développée est la coop avancée, celle-ci s'applique tout d'abord comme le coop de base présenté au-dessus, on l'appel au début de chaque while true pour déterminer un chemin déjà plus optimisé que le astar.

Il utilise le astar pour chaque joueur, une fois le a star appliqué l'algorithme va remplir un dictionnaire, de clé la case, et de valeur la liste des moments ''i'' où elle est occupée par un joueur. Lorsque les case a un moment ''i'' n'est pas prise l'algorithme continue son chemin, il n'y aura aucune collision ''simple'' possible.

Lorsque un joueur veux aller sur la même case, au même moment ''i'' qu'un joueur précédent, on fait un Astar de la case précédente, à la case suivante, en considérant la case prise comme mur, puis on insert ce nouvel Astar a la bonne position, avant de continuer l'algo.

Si jamais la case n'est pas une case prise, on l'ajoute dans le dictionnaire, et si elle est déjà prise à un instant différent, on modifie sa valeur pour garder en compte les instants non disponibles

Malheureusement, cet algo ne règle pas le problème de ''danse'', il n'y a pas moyen avec cet algo de déterminer (facilement) si deux joueurs vont arriver face et face et vouloir aller sur la case de l'autre.

3)Conclusion /Ouverture

Ce projet nous a réellement permis de mieux comprendre les algorithmes de recherche pathfinding et de nous initier au travail sur l'IA. Cela nous aussi permis de comprendre que le plus compliqué n'est pas de trouver le chemin le plus rapide mais de gérer le problème de collision lorsque nous travaillons sur des cas multijoueur, et de patcher ces problèmes.

Nous avons passé beaucoup de temps sur notre Astar et avons pris beaucoup de retard mais avons fini par terminer le projet tout de même. Cependant nous aurions aimé plus nous pencher sur certain problèmes causés par les collisions.

-Tout d'abord les problèmes de danse que nous avons essayé de gérer au mieux malheureusement certain cas existe ou les personnage se bloc pendant quelques itérations et donc crée ce phénomène de ''Danse''.

-Un autre problème est apparu notamment lorsque que nous réduisons la taille de la carte, ce problème est causé par un cas précis, lorsque que deux personnages sont face à face et se bloquent car ils sont sur la fiole de leur adversaire. Ce problème nous a bloqué longtemps et nous ne sommes parvenu à le réglé.

Nous avons laissé en commentaire les morceaux de code sur lesquelles nous avons travaillé.

Merci de votre lecture Thomas et Flavien