KRISNI Almehdi – 3800519
KHARBOUCHE Ibrahim - 3671035

Rapport du Projet LU3IN025 – IA et Jeux

1) Présentation du projet et structure :

Le but de ce projet est de mettre en place des stratégies permettant à des agents de résoudre un problème de déplacement sur une carte. Ce déplacement consiste à partir d'un point initial et de finir sur un point objectif, représenté par un objet sur la carte. On cherche alors à créer des stratégies permettant de réaliser ce déplacement tout en évitant les collisions avec les autres agents présents sur la carte.

Nous avons comme objectif de comparer les différentes stratégies que nous avons codé et de mettre avant le résultat de ces comparaisons, en faisant sorte de trouver la meilleure explication à ces résultats.

Nous allons dans un premier temps présenter les différentes stratégies que nous avons mis en place, puis le système de gestion des collisions entre agents et pour finir ce que nous avons réussir à déduire des comparaisons entre nos algorithmes.

SORBONNE UNIVERSITÉ

KRISNI Almehdi – 3800519
KHARBOUCHE Ibrahim - 3671035

2) Stratégies mises en place :

Nous avons mis en place au total 3 stratégies de résolution différentes :

- Algorithme GreedyBestFirst
- Algorithme RandomBestFirst
- Algorithme Cooperative A*

Nous avons également mis en place des alternatives en implémentant un système de compteur avant que l'agent ne recalcule son chemin.

GreedyBestFirst :

Le principe du GreedyBestFirst est de chercher à partir du nœud initial, soit la case initiale de l'agent au début du problème, le nœud voisin le plus prometteur en terme d'heuristique. L'heuristique utilisée dans le projet est celle de la distance de Manhattan entre une case et l'objectif.

Le fonctionnement de l'algorithme est basé sur un système de nœuds ouverts et de nœuds fermés. Les nœuds ouverts représentent les possibles candidats lors de la sélection du prochain nœud à étudier. Les nœuds fermés représentent les nœuds déjà explorés, et donc n'offrant plus aucune possibilité de recherche.
Il est important de préciser qu'on ne peut pas explorer une nouvelle fois un nœud déjà exploré ce qui permet de ne pas effectuer de boucle lors des recherches.

A chaque itération, on ajoute dans les nœuds ouverts les nœuds voisins (pas encore explorés) du nœud étudié. On cherche ensuite parmi ces nœuds ouverts le nœud le plus prometteur et on répète le procédé jusqu'à ce que le nœud étudié soit l'objectif. On peut alors créer le chemin allant de la case initiale à la case objectif grâce à la parenté des nœuds : on remonte jusqu'à la racine des nœuds.

La particularité de cet algorithme est de renvoyer le premier chemin trouvé. Il ne cherche pas à trouver de meilleurs chemins après avoir trouvé une solution. On possède également la garantie de trouver un chemin entre la case initiale et la case objectif, si ce dernier existe.

RandomBestFirst :

Le principe du RandomBestFirst est le même que celui du GreedyBestFirst à une importante exception près : il ne choisit le prochain nœud ouvert allant être étudié en fonction de l'heuristique la plus optimale, le choix est réalisé aléatoirement.

La particularité de cet algorithme est, comme le GreedyBestFirst, de renvoyer le premier chemin. Le chemin renvoyé a de très grandes chances d'être rempli de mouvements inutiles mais on possède toujours la garantie de trouver un chemin entre la case initiale et la case objectif, si ce chemin existe.

KRISNI Almehdi – 3800519
KHARBOUCHE Ibrahim - 3671035

*Cooperative A** :

On part de l'algorithme A* donné et on ajoute en paramètre un dictionnaire partagé par tous les joueurs de la même équipe qui permet de vérifier si la case dans laquelle l'algorithme veut nous déplacer n'est pas prise par un coéquipier à l'instant t.

L'instant auquel notre joueur passe par la case est retrouvé grâce au coût qui correspond exactement au temps utilisé si on ne prend pas en compte de pauses. Le dictionnaire est d'abord vide et rempli progressivement par chaque joueur, à chaque fois avant d'avancer dans les Nœuds, on vérifie que le dictionnaire ne contient pas la clé (x,y,t) de notre état pour éviter toute collision avec un allié. Si le dictionnaire contient déjà cette entrée, on cherche un chemin alternatif. On peut se retrouver sans aucun chemin jusqu'à l'objectif.

Pour éviter cela, on aurait implémenté un système de pauses : il suffit d'incrémenter notre fonction de calcul du coût de 1, qui représenter l'action de rester sur place, et on le compare avec le coût d'un chemin alternatif pour choisir le meilleur chemin.

On s'est retrouvé au final avec des collisions entre coéquipiers qui survenaient lorsque l'allié a atteint l'objectif et ne bouge plus. Pour éviter cela, il a suffit d'ajouter dans le dictionnaire autant d'entrée que nécessaire pour complètement le remplir de (x,y,t) à (x,y,150), avec x et y les coordonnées de l'objectif du joueur, t le moment où il y arrive (=len(path)-1) et enfin 150 qui est choisie arbitrairement de manière à être supérieur au nombre maximum de tours.

Alternatives avec compte à rebours :

Le but de ces alternatives est de permettre à un agent utilisant cette stratégie de recalculer son chemin vers son objectif lorsque qu'une certaine période de temps s'est écoulée.

Il existe 3 alternatives utilisant ce principe de compte à rebours :

- Alternative de l'algorithme A* donné dans le sujet
- Alternative de l'algorithme GreedyBestFirst
- Alternative de l'algorithme RandomBestFirst

Le fonctionnement du compte à rebours est simple : à chaque déplacement d'un agent utilisant une stratégie faisant appel au compte à rebours, on réduit de 1 le compteur. Une fois le compteur à 0, on recalcule le chemin de l'agent avec la stratégie sélectionnée. Cette stratégie est très efficace lorsque plusieurs collisions entre agents ont lieu lors des déplacements.

Nous allons désormais présenter la manière dont nous avons décidé de gérer les collisions entre agents.

KRISNI Almehdi – 3800519
KHARBOUCHE Ibrahim - 3671035

3) Gestion des collisions :

Nous avons décidé d'implémenter un système de gestion de collisions actif en permanence et de ne pas relier la gestion des collisions à une stratégie particulière. Cela permet d'ajouter une complexité supplémentaire à tous nos algorithmes de recherche.

Nous avons repéré deux situations différentes de collisions : la collision lorsque deux agents vont se déplacer sur la même case, la collision lorsque que deux agents souhaitent permuter de cases et la collision lorsqu'un agent va se déplacer sur une case occupée par un joueur ayant atteint son objectif.

Collision lorsque deux agents vont se déplacer sur la même case :

On se retrouve dans cette situation lorsque deux agents vont se situer la même case après leur prochain déplacement. Afin d'éviter la collision, nous faisons en sorte que le premier agent allant se rendre compte de la collision recalcule immédiatement son itinéraire en utilisant l'algorithme de recherche lui ayant été affecté. Lors du nouveau calcul de l'itinéraire, on considère la case où la collision aurait du normalement avoir lieu comme un obstacle, afin que l'agent ne puisse pas prendre en compte cette case lors de son déplacement.
On note tout de même que cette gestion de la collision défavorise fortement l'agent s'étant rendu compte de la collision en premier puisqu'il devra recalculer son itinéraire tandis que le deuxième agent impliqué dans la collision ne devra pas recalculer son itinéraire puisque le premier agent aura déjà modifié le sien.
On peut éviter les collisions entre coéquipiers en utilisant l'algorithme de recherche Cooperative A* puisqu'il permet de coordonner les mouvements d'agents faisant partie de la même équipe.

Collision lorsque deux agents souhaitent permuter de cases :

Il est mentionné dans les consignes du projet que deux agents ne peuvent pas permuter de case puisqu'il y aurait donc une collision (invisible lors de l'affichage du jeu).
On gère ce cas en faisant en sorte que comme dans le premier cas, l'agent allant se rendre compte en premier de cette collision recalcule son itinéraire, le défavorisant une nouvelle fois.

Collision lorsque qu'un agent rencontre un agent statique :

Nous disposons d'une gestion semi-automatique de ce cas de collision. Lorsqu'un agent est arrivé sur sa case objectif, nous allons considérer sa position comme un obstacle aux yeux de tous les autres agents sur la carte. Ses agents sont considérés comme statiques, puisqu'ils ne bougeront plus de leur emplacement pour le reste du jeu.
La gestion est comme mentionnée précédemment semi-automatique, puisqu'il faudrait que les agents recalculent leurs itinéraires afin qu'ils prennent en compte ce nouvel obstacle.
Nous avons tout de même implémenté une gestion de collision où si un agent devait à entrer en collision avec un agent statique, ce premier recalculerait immédiatement son itinéraire afin d'éviter ce dernier.

KRISNI Almehdi – 3800519
KHARBOUCHE Ibrahim - 3671035

4) Situations intéressantes :

Lors de nos séries de tests, nous avons pu remarquer certaines situations intéressantes. Nous avons décidé d'en présenter quelques-unes.

Pour Cooperative A* :

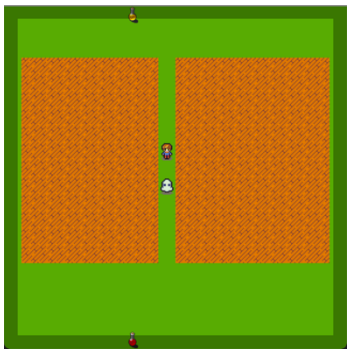
Quand tous les chemins alternatifs sont impossibles, n'ayant pas programmer la pause, le joueur avance jusqu'à être bloqué puis s'arrête jusqu'à la limite de tours.

Pour GreedyBestFirst :

Il est arrivé qu'un joueur se dirige vers son objectif par la droite puis soit bloqué par un allié qui est arrivé à son objectif, soit une collision, il décide alors de faire marche arrière et de faire un détour. La fonctionnalité du compte rebours permet aurait permis à l'agent de recalculer son itinéraire après un certain temps au lieu d'effectuer un détour complet.

Pour les stratégies utilisant le compte à rebours :

Il est arrivé que lorsque deux agents se retrouvent face-à-face dans un long couloir, si les deux agents utilisent une stratégie implémentant la mise à jour de l'itinéraire après un certain temps et que le compte à rebours est faible, on peut remarquer une situation assez drôle où les agents font constamment des allers-retours au sein du couloir. Les deux agents se retrouvent alors bloqués. Afin d'éviter ce problème, on doit augmenter la valeur du compte à rebours.



(Afin d'observer cette situation, il faut se placer sur la carte 'TestMap' et utiliser des stratégies implémentant un compte à rebours pour les deux agents et une valeur de compte à rebours faible.)