LAY Kenny
BOUBRIK Ihsane

Rapport de fin de projet

I. Stratégie greedy best-first

Cette stratégie se base sur l'algorithme greedy best-first qui consiste à choisir les nœuds ayant leur distance à l'objectif la plus faible. Notre fonction ***greedy*** s'occupe donc de calculer un chemin suivant cette condition tout en évitant les cases occupées à l'instant du calcul. A chaque risque de collision, le chemin est recalculé.

II. Path-slicing

Cette stratégie se base sur l'algorithme A* pour le calcul des chemins. Soit la case i, la case actuelle d'un agent. Si à la case i+1 l'agent risque une collision, un nouveau chemin est calculé entre la case i et la case i+1 et est ajouté au chemin initial.
Cependant, le calcul d'un nouveau chemin peut amener à un détour voire à un retour en arrière. De plus, des boucles infinis peuvent se produire entre deux agents essayant de se contourner. Ainsi, cette statégie est loin d'être optimale.

III. A* indépendants

Cette stratégie se base sur l'algorithme A* pour le calcul des chemins, calculés indépendamment au sein d'une équipe avec recalcul des chemins en cas de risque de collision. Notre fonction ***astar_col*** se base sur la fonction ***astar*** donné pour calculer les chemins en évitant les cases occupées à l'instant du calcul.
L'algorithme A* est une amélioration de l'algorithme greedy best-first. De plus, cette stratégie n'oblige pas un retour en arrière. Elle est donc plus efficace que les deux statégies précédentes.

IV. Cooperative A*

Cette stratégie se base sur l'algorithme A* en prenant en compte les chemins que prennent les agents d'une même équipe. Le chemin est toujours recalculé s'il y a un risque de collision entre deux agents d'équipes différentes. Afin de réaliser cette stratégie, on a ajouté un attribut *time* la classe Noeud. Ainsi notre fonction ***coopAstar*** calcule des chemins en évitant les cases occupées par les agents alliés à un instant donné et par les agents ennemis au moment du calcul.
Cette stratégie, essayant de coordonner les mouvements des agents d'une même équipe, se retrouve être plus efficace dans certains cas.

V. Comparaisons

Pour comparer nos stratégies, nous avons lancé notre programme un certain nombre de fois en changeant la stratégie des équipes, avons récupéré temps que mettait une équipe pour gagner et avons fait la moyenne de ces temps. Ces tests ont été effectués avec 5 fps avec une attribution aléatoire des objectifs à chaque exécution. Pour le cas du path-slicing, la moyenne est faite sur les exécutions où il n'y pas eu de boucle infinie. Nous avons donc le tableau suivant :

	Moyenne des temps (Attribution aléatoire)	Temps pour l'attribution des objectifs à l'opposée
Greedy best-first	5.121	6.56
Path-slicing	5.141	Boucle infinie
A* indépendants	5.027	6.56
Cooperative A*	4.531	6.52

Nous remarquons donc que la stratégie cooperative A* est la plus efficace, suivie de la stratégie A* indépendants et de la stratégie greedy best-first. Le path-slicing pouvant donner des boucles infinies est la moins efficace. Ce résultat était prévisible. En effet, l'algorithme A* est une amélioration de l'algorithme greedy best-first et l'algorithme cooperative A* est une amélioration de l'algorithme A*, ayant plus d'informations. Néanmoins, dans de nombreux cas où les agents d'une même équipe ne se croise jamais, l'algorithme cooperative A* et A* agissent à l'identique.

VI. Conclusion

En partant de l'algorithme A*, qui est une très bonne base, et en l'améliorant, nous avons réussi à calculer des chemins pour des agents appartenant à une équipe et affrontant une autre. L'algorithme greedy best-first peut être efficace dans de nombreux cas simples mais l'est moins que l'algorithme A*. Ainsi, l'algorithme A* et ses améliorations restent les stratégies à privilégier ici.