Rapport projet IA

Antoine Toullalan
Rosa Mendas

2021

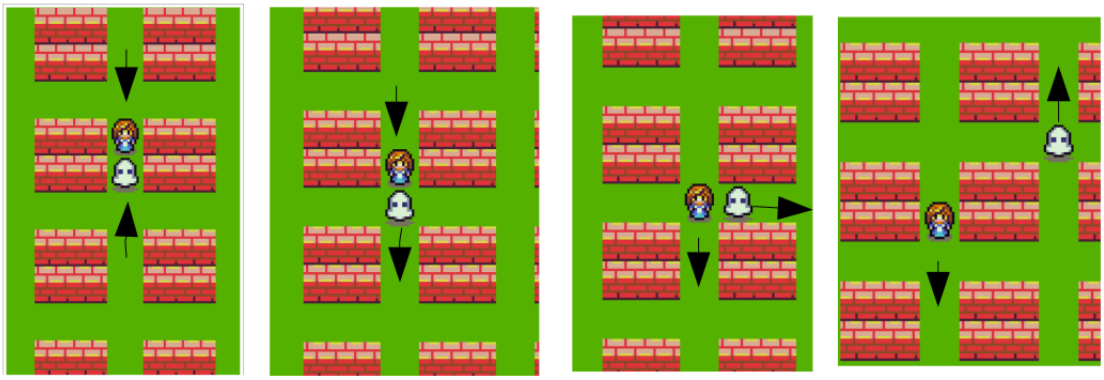

Figure 1: Gestion de collision avec path slicing

Ce projet consiste à développer et comparer plusieurs stratégies pour des équipes d'agents sur une carte avec des obstacles. Chaque agent doit atteindre un objectif sur la carte sans entrer en collision avec un agent de son équipe ou d'une autre équipe. On considère que chaque agent possède un objectif unique et seulement deux équipes. Nous allons comparer les différentes stratégies qui ont été implémentées.

Stratégie 1: Path Slicing

Cette première stratégie implémentée est la plus intuitive: lorsqu'un agent rencontre un obstacle sur son chemin ou atteint un nombre de pas déterminé, il re-calcule une partie de sa trajectoire et retourne sur le chemin initialement calculé.
On a ainsi deux étapes principales :

- chaque agent calcule avec astar(A*) le chemin optimal pour atteindre son objectif sans tenir compte du chemin des autres.
- Lorsqu'un agent voit qu'il va entrer en collision avec un autre agent, il recalcule son chemin pour contourner l'obstacle et retomber sur son chemin n cases plus tard. Si le contour est trop long par rapport au chemin (dans notre programme, si le contour rallonge le chemin initial d'au moins 6 cases) , on recalcule tout le chemin vers l'objectif en contournant l'obstacle. Prenons n=4, on peut se confronter à ce genre de collisions :

Il y a collision à la première image que le fantôme détecte. Il contourne alors l'autre personnage dans les cases suivantes en faisant demi-tour et en tournant à droite.

Stratégie 1 bis: A* avec recalcul du chemin

Contrairement au path slicing, cette stratégie recalcule le chemin entier vers l'objectif. En effet, lors de la détection d'une collision, ou lorsqu'on atteint un certain pas, on ne calcule pas un contour mais un nouveau chemin vers l'objectif. Comme la carte est assez petite, cette stratégie est plus efficace en terme de nombre de coups joué sans augmenter de manière conséquente le temps de calcul. Mais pour une grande carte, on ne peut pas implémenter cette stratégie car ce serait beaucoup trop couteux de recalculer le chemin à chaque collision.

Stratégie 2: Coopérative A*

On monte en complexité et efficacité pour cette stratégie (par rapport à la première) qui est l'implémentation du cooperative path-finding. En effet, dans la première stratégie on pouvait avoir des collisions entre deux personnages d'une même équipe ce qui ralentit les déplacements des personnages. Ainsi on calcule, pour chaque équipe, des chemins qui ne se croisent pas ce qui augmente la rapidité avec laquelle les personnages

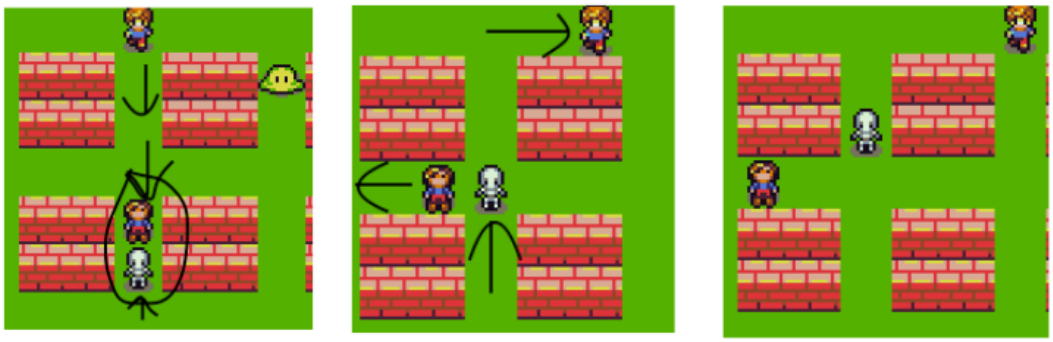

Figure 2: Gestion de collision avec coopérative A*

d'une même équipe atteignent leur objectif.
On suppose que les personnages partagent leurs données (objectifs, chemins calculés, positions) au sein d'une même équipe mais pas avec l'équipe adverse. Pour que les chemins ne se croisent pas , on va introduire la notion de temps dans le calcul des chemins.
Ainsi, au début de la partie, temps=0, l'équipe 1 joue, puis temps=1, l'équipe n2 joue, puis temps=2, l'équipe n1 joue... etc.

Avant que la partie ne commence, on calcule les chemins à l'aide de la classe astar-space-time. Ainsi, aucun agent ne peut occuper la même case (ou croiser un personnage) au même moment qu'un autre agent dans la même équipe.

On vérifie si deux joueurs d'une même équipe possèdent une même position à un même instant donné. Dans ce cas il y a alors collision. Le joueur en mouvement tente par la suite d'éviter ce deuxième en recalculant son chemin grâce à chemin-space-time.
On met à jour la liste des chemins pour la suite de l'algorithme. Lorsqu'un personnage de l'équipe X entre en collision avec un personnage de l'équipe Y, il faut donc que le personnage de l'équipe X recalcule son chemin mais aussi tous les personnages de l'équipe X pour que le path-finding reste coopératif.

Soit John le petit bonhomme en face du squelette. John va détecter la futur collision et recalculer son chemin pour contourner le squelette mais il va aussi indiquer son nouveau chemin aux autres personnages de son équipe. Ainsi on voit le personnage en haut qui est dans la même équipe que John et qui, dans la case 1, se dirigeait vers son objectif en suivant le même chemin que son coequipier. Son chemin va changer afin d'éviter la collision.

Stratégie 3: Alpha-Beta

Cette stratégie est sans doute la plus complexe des stratégie implémentée.
En effet, on calcule l'arbre des possibilités d'une partie en considérant chaque équipe comme un méta-joueur , mais on ne peut pas calculer l'arbre des possibilités pour une partie entière car la complexité de cette stratégie est exponentielle.
On a donc un temps de calcul très long lorsqu'on avance dans la profondeur de l'arbre. On garde donc un arbre de profondeur 3 pour l'implémentation de cette stratégie.

* Initialement, on calcule un arbre de profondeur 3 avec comme racine les positions initiales des personnages, pour la profondeur 1, c'est l'équipe N1 qui joue, on calcule donc toutes les possibilités à jouer pour cette équipe.

* Ensuite les possibilités pour chacun des choix faits pour l'équipe n2, et finalement encore l'équipe n1 pour les choix faits par l'équipe n2.

A chaque feuille, dans cet arbre de possibilités, on assigne un score qui est positif si l'équipe n1 a l'avantage

et négatif si l'équipe n2 a l'avantage. En effet le score est égal au nombre d'objectifs atteints par les personnages de l'équipe n1 fois 1000 moins la somme du nombre de cases à parcourir à priori (avec a-star) pour les personnages de l'équipe n1, le tout moins le nombre d'objectifs atteints par les personnages de l'équipe n2 fois 1000 moins la somme du nombre de cases à parcourir à priori (avec a-star) pour les personnages de l'équipe n2.

L'arbre utilisé est donc un arbre MinMax car l'équipe n1 va chercher à maximiser le score et l'équipe n2 à minimiser le score. On va donc calculer pour chaque équipe son coup qui le favoriser le plus d'après l'arbre de possibilités.
Pour gagner du temps de calcul, on stocke sur une grille de la taille de la carte pour chaque case, la taille des chemins (calculés avec a-star) pour chaque personnage pour qu'il atteigne son objectif, cela permet de ne pas recalculer un score à chaque feuille de l'arbre, et à chaque fois qu'on le met à jour.On se sert de cette grille et des positions des personnages pour calculer le score.
On remet l'arbre de possibilités à jour après chaque coup joué afin qu'on n'atteigne pas le bout de l'arbre après 3 coups. Si au premier coup joué, l 'équipe n1 joue le choix X, alors on étend les feuilles qui ont pour père X.
Notre implémentation de l'arbre de possibilités étant très complexe et codé en un temps assez court, il comporte malheureusement un bug qui est la gestion de collision entre deux personnages de différentes équipes.
Ainsi lorsque deux personnages de différentes équipes se trouvent face à face, le 1er va détecter la collision mais comme il est sur le chemin optimal d'après l'arbre des possibilités, il va choisir de faire pause plutôt que de contourner l'obstacle, l'autre personnage va avoir le même raisonnement et on a la situation où les deux personnages restent face-à-face sans bouger tandis que la partie continue autour d'eux. Ce bug peut être corrigé en re-calculant la grille des scores de telle sorte que, pour les deux personnages impliqués, un des deux contourne l'autre grâce à la nouvelle grille.
Un autre problème de cette implémentation est le temps de calcul pour remettre à jour l'arbre de possibilité, qui est de plusieurs secondes, on a donc visuellement des équipes qui bougent très lentement.

Confrontations des stratégies:

- Path slicing contre A* avec recalcul

Nous avons confronté ces deux stratégies sur différentes cartes afin de voir laquelle performerait le mieux.
Pour chaque carte 100 tests ont été réalisés, voici quelques remarques intéressantes:

- exAdvCoop : Cette carte remplie d'obstacle favorise les collisions. A travers les tests on remarque que les résultats sont proches pour les deux stratégies, mais c'est l'équipe qui choisit le path splicing qui gagne le plus souvent. En effet, la carte étant petite et en mouvement il n'y a pas beaucoup de différence mais il est un peu plus long de calculer tout le path à chaque fois.
- Pour aller plus loin:

Nous avons testé sur une autre carte et les deux stratégies étaient toutes deux démunies face à celle ci. En effet, plusieurs fois un joueur se retrouve coincé entre deux autres joueurs. Le recalcul du chemin si le pas n'est pas assez grand, ne considère pas les joueurs bloquants et calcule indéfiniement les mêmes chemins qui font des allés retours entre les joueurs bloquants.
Les deux stratégies sont souvent ex-aequo.

Pour le reste des stratégies, nous faisant affronter les deux équipes avec la même algoritme en regardant la longueur moyenne des personnages pour arriver à leur objectif.
Pour chaque stratégie, on répète cette opération 10 fois et on a les résultats suivants :

- Stratégie 1 : chemin de longueur moyenne de 16.633 et tous les personnages arrivent à leur objectifs.
- Stratégie 2 : chemin de longueur moyenne de 14.266, et tous les personnages arrivent à leur objectifs.
- Stratégie 3 : chemin de longueur moyenne de 24.8 pour les personnages qui arrivent aux objectifs (environ les 2/3).

Figure 3: Gestion de collision pour aller plus loin

Les mauvaises performances de la stratégie 3 peuvent s'expliquer par la mauvaise gestion des collisions (à cause des bugs).
On remarque que la stratégie 2 est à priori la plus performante car les personnages arrivent plus rapidement à leurs objectifs. La stratégie 2 est plus performante que la 1, en effet il y a moins de collisions à gérer dans le cooperative path-finding qu'avec le path-slicing, donc moins de contour et un chemin plus court. La stratégie la plus performante semble être le cooperative A*. Mais la stratégie avec l'arbre de possibilités à beaucoup de potentiel, il faudrait la débarasser des bugs mais la complexité de l'implémentation et le temps limité imposé ne nous a pas permis de faire mieux.