Sorbonne Université
MASTER ANDROIDE M1 – RP

Mars 2024

Projet de résolution de problèmes : Algorithme A* pour le jeu Ricochet Robots

Introduction et objectifs

L'objet de ce projet est de développer et tester des méthodes de recherche arborescente pour la résolution du jeu Ricochet Robots. Ricochet Robots est un jeu de plateau pour une personne ou plus, conçu par Alex Randolph en 1999. Le principe général du jeu est de déplacer des robots vers des emplacements sélectionnés tout en respectant les limites strictes imposées aux mouvements des robots.

Le jeu est constitué d'un plateau de 256 cases (correspondant à un carré composé de 16 x 16 cases) et de cinq pions appelés "robots" (un bleu, un jaune, un noir, un rouge et un vert). Dix-sept cases du plateau représentent des cases "buts". Chaque case but est identifiée par un symbole (quatre symboles possibles) et une couleur (quatre couleurs possibles : bleu, jaune, noir et bleu). Il existe également une case but spéciale multicolore. Une partie est divisée en tours de jeu. Lors d'un tour de jeu, une case but est sélectionnée au hasard. Les cinq robots sont également positionnés aléatoirement sur le plateau. L'objectif est de déplacer le robot correspondant à la couleur de la case but sélectionnée, sur cette case but. Par exemple, si une case but rouge symbolisant un triangle a été tirée au sort, le but est d'envoyer le robot rouge sur cette case. Si la case but multicolore a été sélectionnée, on peut utiliser n'importe quel robot pour atteindre cette case. Notez qu'il n'existe pas de case but de couleur noire, la seule case but que le robot noir peut donc atteindre est la case multicolore. Les robots se déplacent en ligne droite (vers le haut, vers le bas, vers la droite ou vers la gauche) et ne peuvent s'arrêter que lorsqu'ils rencontrent un mur ou un autre robot. Tous les robots peuvent être déplacés, chaque déplacement a un coût unitaire (quelle que soit la distance parcourue) et le but du jeu est de trouver une séquence de déplacements des robots (un seul robot est déplacé à la fois) permettant d'amener le robot associé à la couleur de la case but (ou n'importe quel robot dans le cas de la case but multicolore) vers la case but.

Un exemple est donné en Figures 1 et 2 [2].

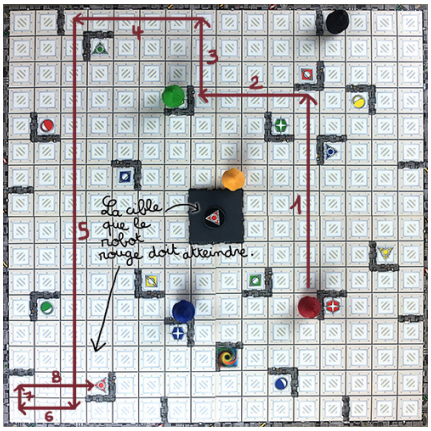


Figure 1: Dans cet exemple, le robot rouge doit atteindre la cible rouge en forme de triangle. Il existe une solution en huit mouvements, où uniquement le robot rouge est déplacé.

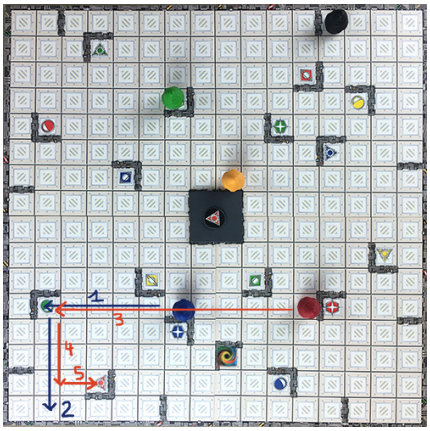


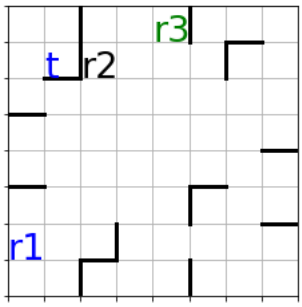
Figure 2: Pour ce même exemple, il existe néanmoins une solution en cinq mouvements : le robot bleu est préalablement déplacé, ce qui permet au robot rouge de prendre appui sur celui-ci.

Ce jeu peut être joué aussi bien en solitaire qu'avec plusieurs joueurs. À plusieurs joueurs, dès que l'un des joueurs a trouvé une solution, il énonce à voix haute le nombre total de mouvements utilisés et retourne un sablier. Ensuite, chaque joueur a la possibilité de trouver une meilleure solution, jusqu'à ce que le temps soit écoulé. Le joueur qui a trouvé le plus petit nombre de mouvements présente sa solution. Si sa solution est correcte, il gagne le tour. Dans le cas contraire, le joueur restant dont le nombre de coups est le moins élevé peut essayer, et ainsi de suite jusqu'à ce qu'un vainqueur soit déclaré. Un nouveau tour commence ensuite : une nouvelle case but est tirée au sort. Lorsque 17 tours ont été réalisés, le joueur qui en a gagné le plus grand nombre de tours gagne la partie.

Dans ce projet, on s'intéresse à réaliser un programme qui, pour un tour donné, est capable de trouver une solution optimale (c'est-à-dire minimisant le nombre de déplacements des robots) ou de déterminer qu'il n'existe pas de solutions de coût inférieur à une certaine valeur m fixée (Ricochet Robots est un problème NP-difficile [1], il ne sera donc pas possible de résoudre toutes les instances de grande taille) ou encore de déterminer qu'il n'existe pas de solutions.

Partie 1 : Modélisation, instances et résolution par recherche arborescente

Un plateau de jeu peut être modélisé par une matrice binaire représentant les murs verticaux, une matrice binaire représentant les murs horizontaux et une matrice d'entiers représentant les robots et les cibles. Un exemple est donné à la Figure 3 pour un plateau simplifié de 8 x 8 cases, comprenant trois robots (r1, r2 et r3, notés 1, 2 et 3 dans la matrice des cases du jeu) et une cible (symbolisée par la lettre "t", notée -1 dans la matrice des cases du jeu).




```
murs verticaux =
[[0 1 0 0 1 0 0]
 [0 1 0 0 0 1 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0]
 [0 0 1 0 0 0 0]
 [0 1 0 0 1 0 0]]

murs horizontaux =
[[0 0 0 0 0 0 1 0]
 [0 1 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [1 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0]]

cases du jeu =
[[ 0  0  0  0  3  0  0  0]
 [ 0 -1  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]]
```

Figure 3: Exemple de modélisation du plateau de jeu

Un code Python est disponible sur Moodle, permettant de générer des instances du jeu. La fonction de génération aléatoire est la suivante : `generateRandomInstance(n,k)` avec n la taille d'un côté du plateau (entier pair supérieur ou égal à 4), et k le nombre de robots (compris entre 1 et 5). Le robot arbitrairement choisi pour atteindre la cible est toujours le robot bleu. On ne considère pas de cible multicolore. Une fonction de visualisation d'un plateau de jeu vous est également fournie.

Question 1. Préciser la représentation des états et indiquer l'état final.

Question 2. Pour un plateau de taille n x n et k robots, donner une borne supérieure de la taille de l'espace d'états.

Question 3. Donner une borne supérieure sur le nombre de successeurs possibles d'un état.

Question 4. Dans un premier temps, nous allons essayer de déterminer une borne supérieure de l'évaluation de la solution optimale. Pour cela, seul le robot bleu (celui devant atteindre la cible) sera déplacé. Implanter une procédure arborescente de recherche dans le graphe d'états afin de trouver une solution optimale dans ce cas particulier. Procéder par recherche en largeur ou en profondeur (au choix). Afin d'éviter que la recherche arborescente ne soit trop longue, vous pourrez arrêter la recherche s'il n'existe pas de solutions de coût inférieur à une certaine valeur m (vous adapterez cette valeur en fonction de la taille de l'instance, du temps de résolution de votre méthode et de votre taille de mémoire RAM).

Question 5. Tous les robots sont maintenant pris en compte. Implanter une procédure arborescente de recherche (en largeur ou profondeur, au choix) dans le graphe d'états afin de trouver une solution optimale. Utiliser la borne supérieure trouvée à la Question 4 afin limiter la taille de l'exploration.

Question 6. Pour une instance de votre choix, illustrer graphiquement la solution optimale obtenue.

Question 7. Donner les temps moyens de la méthode sur 20 instances de taille $n = 8$. Étudier ensuite l'évolution du temps moyen de résolution lorsque n augmente (par pas de 2, jusqu'à ce que le temps moyen devienne supérieur à 10 minutes ou que la mémoire RAM de votre ordinateur sature). Étudier également l'impact du nombre k de robots sur la valeur de la solution optimale.

Partie 2 : Résolution par A*

Dans cette seconde partie, vous allez étudier une procédure A*.

La première heuristique, notée h_1, est la suivante : si le robot et la cible se situent sur la même ligne ou la même colonne, l'évaluation est de 1. Si le robot et la cible se situent sur des lignes et colonnes différentes, l'évaluation est de 2. Enfin, si le robot et la cible sont à la même position (même ligne et même colonne), l'évaluation est de 0.

Question 8. Cette heuristique est-elle minorante ? monotone ? coïncidente ?

Question 9. Implanter A* en considérant l'heuristique h_1. Comparer les résultats avec ceux obtenus en Partie 1.

La deuxième heuristique, notée h_2, est la suivante : on relâche la contrainte stipulant que le robot devant atteindre la cible ne peut pas s'arrêter en cours de déplacement. Une fois un déplacement amorcé, le robot peut donc s'arrêter sur n'importe quelle case (mais il doit toujours s'arrêter s'il rencontre un mur). Les autres robots ne sont pas considérées dans l'évaluation de cette heuristique.

Cette heuristique est illustrée sur l'instance de la Figure 3 à la Figure 4. Partant de la position du robot bleu, on évalue à 1 toutes les positions que peut atteindre le robot avant de rencontrer un mur. On réitère ensuite le processus, à partir de chacune des positions trouvées, en incrémentant l'évaluation d'une unité.

Notez qu'il peut être intéressant de pré-calculer les valeurs de h_2 pour toutes les positions possibles du robot bleu.

Question 10. Implanter A* en considérant l'heuristique h_2. Comparer les résultats avec ceux obtenus en Partie 1 et ceux obtenus avec l'heuristique h_1.

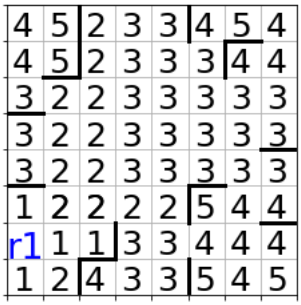


Figure 4: A* : illustration du calcul de l'heuristique h_2

Partie 3 : Résolution par A* bidirectionnel (optionnel)

Dans cette partie optionnelle, il vous est demandé d'étudier et d'implanter une procédure A* bidirectionnelle (en plus du robot bleu devant atteindre la cible, on considère un robot artificiel partant de la cible devant atteindre la position initiale du robot bleu). Comparer les résultats de cette méthode avec la procédure A* de la Partie 2.

Organisation et dates

- Le langage d'implantation recommandé est Python, toutefois, si ce langage ne vous convient pas, d'autres options sont éventuellement admissibles (Java ou C/C++).
- Les projets doivent s'effectuer en binôme. Informez votre chargé de TD des binômes constitués dès que possible (thibaut.lust@lip6.fr). En cas de difficulté à constituer un binôme, une dérogation pourra être accordée à titre exceptionnel après demande motivée (par e-mail) auprès du chargé de TD avec cc au responsable du projet (patrice.perny@lip6.fr).
- Le projet doit être rendu au plus tard le **22 avril 2022** à 23h59. Votre livraison sera constituée d'une archive ZIP qui comportera les sources du programme (avec les instructions pour l'exécution) et un rapport au format PDF (de préférence rédigé en LaTeX). Le plan du rapport suivra le plan du sujet et résumera les choix méthodologiques, les principales réalisations et les tests numériques. L'archive ZIP, dont le nom contiendra les noms des deux personnes constituant le binôme, devra être soumise sur Moodle.
- La soutenance du projet est prévue en salle TME le 25 avril 2024. Prévoyez de pouvoir faire une démonstration sur machine de vos programmes.

References

[1] Birgit Engels and Tom Kamphans. Randolphs robot game is NP-hard! *Electronic Notes in Discrete Mathematics*, 25:49–53, 2006.

[2] UnMondeDeJeux. Et si on jouait à ricochet robots. `https://www.lemonde.fr/blog/unmondedejeux/2016/05/28/et-si-on-jouait-a-ricochet-robots/`, 2016.