15611 EHRET Augustin

Étude et résolution du jeu 'Ricochet Robots'

'Ricochet Robots' est un jeu de plateau unique en son genre. Il confronte la performance de chacun face à un problème simple : amener l'un des 4 robots vers un état final, à l'aide des autres.

On peut se demander comment traiter ce problème de manière algorithmique et efficace.

Via la conversion du jeu en une interface informatique, on cherche, étant donné un plateau initial, à le transformer en une position finale voulue. Cette transformation repose sur une succession de transitions d'états des robots.

Positionnement thématique (ÉTAPE 1) :

- INFORMATIQUE (Informatique pratique)
- INFORMATIQUE (Informatique Théorique)

Mots-clés (ÉTAPE 1) :

Mots-clés (en français)	**Mots-clés (en anglais)**
Algorithme de Dijkstra	*Dijkstra algorithm*
*Algorithme A**	*A* algorithm*
graphe pondéré connexe	*connected weighted graph*
parcours en largeur	*Breadth-First Search*
Optimisation	*Optimization*

Bibliographie commentée

Ricochet robots se joue sur un plateau quadrillé avec des obstacles et 4 robots. Chaque manche d'une partie est plus ou moins similaire : les joueurs doivent trouver comment emmener un des 4 robots sur un point précis du plateau, avec pour seule règle que le robot s'arrête à chaque obstacle rencontré, et cela, en le moins de coups possible. Il s'agit donc là recherche de plus court chemin.

En modélisant alors le problème sous forme d'un graphe où les sommets représentent une configuration des robots[1], on se ramène à un problème de plus court chemin dans un graphe, problème bien étudié de la théorie des graphes. Il existe en revanche de nombreuses variantes

de sa résolution, dépendant notamment de la nature et les caractéristiques du graphe en question, mais nous reviendrons plus tard sur le cas de Ricochet Robots.

Dans le domaine de la recherche du plus court chemin, un tournant majeur a été franchi en 1959 avec la proposition d'un algorithme par E. W. Dijkstra. Cet algorithme permet de déterminer le plus court chemin entre deux sommets d'un graphe connexe pondéré [2], posant ainsi les bases d'une méthodologie qui sera largement utilisée dans divers domaines. Un peu plus tard, en 1968, Peter E. Hart, Nils John Nilsson et Bertram Raphael introduisirent l' algorithme A*, qui constitue une extension significative de celui de Dijkstra. L'algorithme A* optimise l'exploration du graphe en priorisant les états se trouvant à proximité de la cible, grâce à l'utilisation d'une fonction heuristique [3]. Cette approche permet de réduire le facteur de branchement moyen, rendant ainsi la recherche plus efficace.

Dans un environnement comme celui de Ricochet Robots, la prise en compte de la complexité est cruciale et le malgré le coût en exactitude, une approche heuristique est plus que nécessaire. En effet, le jeu se caractérise par un espace d'états explosif : avec 4 robots sur un plateau 16x16, le nombre de configurations dépasse 10^9. Cela est un défi même pour un ordinateur. Pour illustrer : si chaque configuration était stockée avec 64 bits, l'ensemble occuperait environ 5,5 Go de mémoire. Pour un ordinateur (même avec 8 ou 16 Go de RAM), cela est considérable. De plus, la nature coopérative des robots introduit une couche de complexité supplémentaire. Un mouvement peut libérer un chemin pour un autre robot, nécessitant une modélisation des états incluant toutes les positions robotiques simultanées.

Pour garantir l'optimisation du nombre de mouvements effectués, un parcours en largeur (BFS) est ainsi employé.[4] Cette approche permet d'explorer toutes les configurations accessibles à partir de l'état initial, niveau par niveau, assurant ainsi que dès que la position cible est atteinte, le nombre minimal de coups a été réalisé. Néanmoins, dans une approche naïve, vérifier si une configuration a déjà été visitée impliquerait une recherche linéaire dans une liste, avec une complexité en O(n), ce qui n'est pas viable compte tenu du nombre de configurations possibles.

Pour contourner ce problème, une hybridation entre le BFS et l'algorithme A* est mis en place. L' introduction des tables de hachage permet de réduire la vérification d'une configuration déjà visitée à une recherche en O(1) dans le cas idéal, ce qui est primordial vu le nombre de sommet possible dans notre représentation (Pour 4 robots dans un plateau de taille 16x16, le nombre de sommets est 699 170 560) [5]

Pour cette étude et valider nos hypothèses ainsi que nos résultats, il est indispensable de disposer d'un outil de test efficace. C'est pourquoi l'interface de programmation Pygame est particulièrement adaptée pour travailler sur les déplacements des robots. [6]En effet l'interface interactive proposée permet de visualiser en temps réel l'évolution des configurations. Elle permet également d'analyser clairement les éventuels problèmes rencontrés lors de l' implémentation de l'algorithme, et même nous le verrons, de jouer au jeu.

Problématique retenue

Est-il possible de résoudre informatiquement une partie de Ricochet Robots en un temps limité, malgré l'explosion combinatoire des états ?

Objectifs du TIPE du candidat

1. Modéliser le jeu en graphe orienté pondéré.

2. Implémentation du BFS pour trouver le plus court chemin

3. Optimisation de la gestion mémoire via des structures de hachage.

4. Développement d'une interface interactive via Pygame.

Références bibliographiques (ÉTAPE 1)

[1] CONCOURS CENTRALE : Étude du jeu Ricochet Robots *: https://a.www.concours-centrale-supelec.fr/CentraleSupelec/2018/MP/sujets/N001.pdf*

[2] E. W. DIJKSTRA : A short introduction to the art of programming *: https://www.cs.utexas.edu/~EWD/ewd03xx/EWD316.PDF*

[3] P. E. HART : A Formal Basis for the Heuristic Determination of Minimum Cost Paths *: https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf*

[4] F. KARDOŠ : Algorithmique des graphes – Parcours en largeur *: https://www.labri.fr/perso/fkardos/cours3a.pdf*

[5] ADAM KIRSCH, MICHAEL MITZENMACHER, AND GEORGE VARGHESE : Hash-Based Techniques for High-Speed Packet Processing *: https://www.eecs.harvard.edu/~michaelm/postscripts/dimacs-chapter-08.pdf*

[6] : Pygame Documentation *: https://www.pygame.org/docs/*