Ecole Publique d'Ingénieurs en 3 ans

Rapport

Project: Rational Agent in Wumpus World

le 09 Janvier 2023,
version 1.1

PATRY alan && CHEDDAD saad | **Professeur tuteur : Loïc Simon**

TABLE DES MATIERES

Table des matières

1. Introduction

Le but du jeu Hunt the Wumpus est d'explorer une grotte remplie de dangers, tels que des gouffres sans fond et le Wumpus, un monstre qui mange toute personne qui entre dans sa salle. L'objectif est de trouver une pile d'or dans la grotte et de retourner à la case de départ avec le meilleur score possible. L'agent peut percevoir son environnement grâce à divers indices, tels que la puanteur du Wumpus, les brises dégagées par les gouffres, le scintillement émis par l'or et le cri du Wumpus à sa mort. Il peut également effectuer des actions telles que se déplacer, tirer une flèche, collecter de l'or et quitter la grotte. La tâche consiste à concevoir un agent rationnel capable de naviguer efficacement dans cet environnement inconnu.

2. Stratégie

L’agent est placé sur une grille sans connaitre son contenu. Son objectif principal est de trouver l’or, de tuer le Wumpus et de retourner à sa position de départ pour sortir.

2.1. L’algorithme A*

2.1.1. Implémentation

L'algorithme A* a été utilisé dans ce programme de chasse au Wumpus pour trouver le chemin le plus court vers l'objectif en utilisant une heuristique pour orienter la recherche. Pour ce faire, l'algorithme a commencé par initialiser une file de priorité contenant l'état initial avec un coût calculé en fonction de l'heuristique de l'état et du nombre d'états explorés jusqu'à présent. Ensuite, une boucle while a été créée pour itérer tant que la file de priorité n'est pas vide. À chaque itération, l'algorithme a récupéré l'état de coût le plus faible de la file de priorité, l'a marqué comme visité et a vérifié si l'objectif a été atteint. Si ce n'est pas le case, de nouveaux états ont été générés en appelant la fonction “generate_next_states” , et leur coût total a été calculé en ajoutant la longueur du chemin actuel à l'heuristique de l'état suivant. Les nouveaux états ont été ajoutés à la file de priorité en utilisant leur coût total comme priorité. Si la boucle while se termine sans que l'objectif n'ait été atteint, cela signifie que la file de priorité est vide et qu'aucun chemin vers l'objectif n'a été trouvé. Dans ce cas, un message indiquant qu'aucune solution n'a été trouvée a été affiché et une liste vide a été renvoyée.

2.1.2. Exploration et exécution

La fonction “think” à deux phases principales : l'exploration et l'exécution. Pendant la phase d'exploration, l’Agent cherche à apprendre davantage sur son environnement et à trouver des plans d'action pour atteindre ses objectifs. Pendant la phase d'exécution, l'Agent suit un plan d'action pour atteindre son objectif :

- ✓ Si l'Agent détecte de l'or, il exécute immédiatement l'action “grab” pour prendre l'or.
- ✓ Si l'Agent a déjà pris l'or et se trouve sur la cellule (1,1), il exécute immédiatement l'action “climb” pour sortir de la caverne.
- ✓ Si l'Agent connaît l'emplacement du Wumpus, elle utilise la fonction solve pour trouver le chemin le plus court pour le tuer et stocke les actions à exécuter dans la liste “next_actions”.
- ✓ Si l'Agent a déjà pris l'or, tué le Wumpus et qu’il est en train d'apprendre (isLearning est True), il utilise la fonction solve pour trouver le chemin le plus court pour sortir de la caverne et stocke les actions à exécuter dans la liste “exitPath”. Il arrête donc d’apprendre : cette liste n’est pas mis-à-jour pour éviter des calculs inutiles, puisque l’agent utilise uniquement des cases sûres pour sortir il n’est pas nécessaire de recalculer le chemin.
- ✓ Si l'Agent n'a pas encore exploré la carte de manière sûre et qu'il est en train d'apprendre (isLearning est True), il utilise la fonction solve pour trouver le chemin le plus court vers une cellule “safe” non explorée et stocke les actions à exécuter dans la liste “next_actions”.
- ✓ Enfin si l’agent n’a plus de case “safe” à explorer et que l'objectif principal n’est toujours pas rempli, alors il tente des cases non sûres (PITP, WUMPUSP) en se basant sur un indice de danger induit par chaque case adjacente. Malheureusement il n’est parfois pas possible de différencier un faux positif d’un vrai positif, l’algorithme s’en remet donc entièrement à la chance quand plusieurs cases ont un indice équivalent.

La condition d’arrêt d’A* en phase d’exploration était à l’origine que toute les cellules “safe” soient explorées, mais à cause de temps de calcul trop longs lorsque de nombreuses cellules “safe” sont à explorer, nous avons été contraints de relaxer cette condition à l’exploration de la cellule “safe” la moins couteuse, le chemin d’exploration perd donc en optimalité puisqu’il arrive à l’agent de faire des détours peu judicieux.
De manière analogue, le nombre de chemin disponible lors du calcul du retour étant souvent assez élevé nous avons valoriser l’action “forward” de sorte à privilégier les lignes droites.

2.1.3. Heuristique

Notre heuristique calcul la valeur d'un état en se basant sur l'objectif en cours, par, exemple, lorsque l'agent explore prudemment la map ou quand il essaye de sortir de la cave, nous ne voulons pas que l'agent prenne le moindre risque en allant sur une case potentiellement dangereuse. De la même manière en phase d'exploration et donc de recherche de l'or, ne connaissant pas sa position il est impossible de compter la distance de Manhattan, en revanche lors du calcul du chemin de sortie, cela devient pertinent.

2.1.4. Génération des états suivants

La génération des états suivant a été pensée pour ne pas générer d'états superflus ou redondants, il est donc par exemple impossible de générer les actions "climb", "kill" ou "grab" sans que les conditions de leurs utilisations ne soient remplies. De même si le dernier mouvement est une rotation à droite ou à gauche le prochain ne sera jamais l'inverse, pareil pour les ¾ de tour équivalent à -¼ de tour. Tout ceci nous permet de considérablement réduire le temps de calcul d'A*, sans perdre en optimalité.

3. Conclusion

En résumé, Ce projet consistait à résoudre un problème de recherche en utilisant un agent autonome pour parcourir des cartes de différentes tailles. Notre agent est capable de compléter la plupart des cartes dans un temps raisonnable et fourni une solution personnalisée au problème. Cependant, il y avait de nombreuses contraintes à prendre en compte lors de la résolution du problème, comme les considérations d'optimisation temporelle et physique. Ce projet nous a permis de mieux appréhender la métodologie d'analyse d'un problème et les étapes de sa résolution, tout en mettant en lumière comment de petits détails peuvent avoir un impact significatif sur la solution et son temps de calcul. De plus nous avons pu mettre en pratique les compétences acquises lors des séances de cours et des travaux pratiques dans un cadre non contrôlé (algorithme non défini par les encadrants) et de manière ludique. Nous avons eu l'occasion de mettre en œuvre différentes techniques de recherche, telles que l'algorithme A*, DFS, BFS, IDS, comparer leurs résultats et développer une stratégie pour résoudre le problème du jeu Hunt the Wumpus.

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche

Normandie Université

Cti Commission des titres d'ingénieur

cdefi Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES