RAPPORT DE PROJET MARC ROVER

INTRODUCTION

Dans le cadre de ce projet, nous avons dû réaliser la manipulation d'un robot virtuel, Marc Rover, pour explorer une carte en utilisant des structures de données avancées en C. Ce projet vise à modéliser et manipuler un système de navigation qui permet à un rover de se déplacer à travers une grille représentant un environnement planétaire.

Pour ce projet, nous avons conçu plusieurs structures essentielles, notamment Node et Tree. Chaque Node représente une position du rover sur la carte, incluant le type de terrain, le coût de la navigation et les mouvements disponibles. La structure Tree permet d'organiser ces noeuds pour modéliser les différents chemins possibles que le rover peut emprunter à partir de sa position initiale. Le projet se concentre également sur la gestion de la mémoire, étant donné que les nœuds sont alloués dynamiquement lors de la construction de l'arbre.

L'objectif final est de trouver le chemin le plus efficace jusqu'à la base, en évitant les obstacles et en minimisant le coût des déplacements.

DIFFICULTÉS DU PROJET

Dans ce projet, nous avons manipulé des structures de données dynamiques complexes pour la représentation des différents chemins du rover. Plusieurs défis sont apparus lors de l'utilisation des structures d'arbre pour modéliser le parcours de Marc Rover.

- Gestion de la Mémoire : La mémoire a été allouée dynamiquement pour chaque noeud de l'arbre, ce qui a introduit la possibilité de fuites de mémoire. Chaque noeud créé devait être libéré correctement pour prévenir les problèmes de gestion de la mémoire. Cela a demandé une attention particulière lors de la libération des ressources allouées.

- Gestion des Pointeurs : La gestion des pointeurs était essentielle pour préserver les relations entre les différents nœuds de l'arbre. L'arbre étant composé de plusieurs fils à chaque noeud,

il était crucial de maintenir une bonne référence des adresses de chaque noeud pour éviter de perdre des branches entières de l'arbre.

- Profondeur et Complexité de l'Arbre : La création d'un arbre qui permet de modéliser toutes les possibilités de mouvement de Marc Rover en fonction des contraintes de la carte (obstacles, terrains impraticables) a entraîné une complexité qui a été difficile à gérer. Nous avons dû limiter la profondeur de l'arbre pour éviter une explosion combinatoire.

MODÉLISATION

- ***Utilisation d'un Arbre pour les Trajets***

Avantages :

- Permet de modéliser tous les chemins possibles à partir de la position initiale.

- Accès rapide à chaque noeud de l'arbre pour des opérations de recherche.

Inconvénients :

- Problèmes de mémoire : Étant donné la taille potentielle de l'arbre, la mémoire peut être vite saturée.

- Gestion complexe des pointeurs : Une erreur dans la manipulation des pointeurs pourrait entraîner la perte de l'accès à une grande partie de l'arbre.

Fonctionnalités demandées :

- F1 : Création de l'Arbre

Création d'un arbre de décision à partir de la position initiale du rover en modélisant chaque mouvement possible comme un noeud enfant. Chaque noeud représente une nouvelle position du rover sur la carte.

- F2 : Recherche du Chemin Optimal vers la Base

Nous avons implémenté une fonction `pathToBase()` qui recherche le chemin ayant le coût minimal pour atteindre la base. Cela a été fait en utilisant une recherche en profondeur sur l'arbre et en gardant une trace des chemins avec le coût le plus bas.

Avantages :

- Gestion simplifiée des déplacements : La file permet de stocker les positions à visiter et de suivre un ordre logique pour explorer chaque noeud.

- Optimisation de la recherche : Les piles ont été utilisées pour garder une trace des chemins parcourus lors de la recherche de la base.

Inconvénients :

- Gestion de la mémoire : Chaque position ajoutée à la file ou à la pile nécessite une allocation dynamique, ce qui peut être sujet aux fuites de mémoire si elles ne sont pas correctement libérées.

- F3 : Utilisation d'une File pour les Positions à Visiter

Nous avons utilisé une file pour garder une trace des positions à explorer lors de la recherche du chemin vers la base. Cette approche permet d'explorer les positions de manière ordonnée.

- F4 : Utilisation d'une Pile pour le Chemin Actuel

Une pile a été utilisée pour sauvegarder le chemin parcouru jusqu'à atteindre la base, facilitant ainsi la reconstruction du chemin optimal.

FONCTIONNALITÉS DÉTAILLÉES

F1 : Création de l'Arbre de Décision

La fonction `createTreeRecursivity()` crée l'arbre de décision à partir de la position initiale du rover. Chaque noeud représente une nouvelle position en fonction des mouvements possibles (avant, arrière, gauche, droite). L'arbre est créé de manière récursive jusqu'à atteindre une certaine profondeur ou une position invalide.

F2 : Recherche du Chemin vers la Base

La fonction `pathToBase()` permet de trouver le chemin ayant le coût minimal vers la base. Cette fonction parcourt l'arbre de manière récursive et compare les coûts des différents chemins pour sélectionner le chemin optimal.

F3 : Enqueue et Dequeue

Nous avons implémenté les fonctions `enqueue()` et `dequeue()` pour manipuler les files utilisées dans le cadre du projet. Ces fonctions nous ont permis de suivre les positions à visiter de manière efficace.

CONCLUSION

Ce projet a permis de mettre en pratique plusieurs concepts clés de la programmation en C, notamment la gestion de la mémoire dynamique, l'utilisation des structures d'arbre, de files, et de piles. La principale difficulté résidait dans la gestion des pointeurs et la prévention des fuites mémoire. En utilisant des structures de données adaptées, nous avons été en mesure de modéliser efficacement les différents chemins possibles pour Marc Rover.

L'étude de la modélisation de l'arbre de décision et de l'optimisation du chemin vers la base nous a fourni une base solide pour comprendre l'importance de la complexité algorithmique et de la gestion des ressources en programmation embarquée et en robotique.

PERSPECTIVES

Pour les futures évolutions de ce projet, nous pourrions considérer l'intégration d'algorithmes plus avancés pour l'optimisation des chemins, tels que l'algorithme A*, ainsi qu'une meilleure gestion de la mémoire pour minimiser les risques de fuites lors de l'exploration de grands espaces.