

DOCUMENTATION:

Bonnes pratiques en Rust

- Utiliser la propriété et le système de typage de Rust pour garantir la sécurité de la mémoire et la robustesse du code.
 - Préférer les types immuables whenever possible.
 - Utiliser des références et des pointeurs intelligents de manière appropriée.
 - Éviter les erreurs de nullité.
- Profiter des fonctionnalités de sécurité de la mémoire de Rust, telles que la vérification des limites de tableau et la détection des fuites de mémoire.
- Utiliser des bibliothèques Rust standard et éprouvées pour les tâches courantes, telles que la gestion des entrées/sorties, la sérialisation et la concurrence.
- Écrire des tests unitaires complets pour s'assurer que le code fonctionne comme prévu.
- Documenter le code de manière appropriée à l'aide de commentaires et de documentation générée automatiquement.

Outils pour améliorer la qualité du code

- `rustfmt`: Formate le code Rust selon un style cohérent.
- `cargo test`: Exécute les tests unitaires pour le code Rust.
- `clippy`: Outil de linting pour identifier les problèmes potentiels dans le code Rust.
- `rustdoc`: Génère une documentation à partir des commentaires du code Rust.

L'algorithme de Dijkstra

Inventé par le scientifique informatique néerlandais Edsger Dijkstra en 1959, est un algorithme de recherche de chemin célèbre en informatique. Il permet de trouver le chemin le plus court entre deux points dans un graphe, c'est-à-dire un réseau d'éléments (appelés nœuds) reliés par des liens (appelés arêtes).

Comment fonctionne-t-il ?

Imaginez un labyrinthe et que vous cherchez le chemin le plus court pour sortir. L'algorithme de Dijkstra fonctionne de manière similaire :

1. **Départ:** On commence par le nœud de départ et on le marque comme "visité".
2. **Exploration:** On explore ensuite tous les nœuds voisins du nœud de départ, en calculant pour chacun la distance totale parcourue (en ajoutant le coût de l'arête au coût cumulé jusqu'au nœud de départ).
3. **Choix:** On choisit ensuite le nœud non visité avec la plus petite distance totale calculée, et on le marque comme "visité".
4. **Répétition:** On répète les étapes 2 et 3 jusqu'à atteindre le nœud d'arrivée.

En résumé, l'algorithme de Dijkstra construit progressivement le chemin le plus court en explorant les nœuds étape par étape, en tenant compte des coûts des arêtes.

Pourquoi est-il utile ?

L'algorithme de Dijkstra trouve de nombreuses applications dans des domaines variés, notamment :

- **Routage:** Pour trouver les meilleurs itinéraires sur les cartes routières ou les réseaux informatiques.
- **Planification:** Pour optimiser les déplacements de robots ou de véhicules autonomes.
- **Logistique:** Pour organiser efficacement les livraisons et les flux de marchandises.
- **Jeux:** Pour trouver les chemins les plus courts dans les jeux vidéo ou les puzzles.

Avantages:

- **Simplicité:** L'algorithme est relativement simple à comprendre et à implémenter.
- **Efficacité:** Il trouve généralement le chemin le plus court de manière rapide et efficace.
- **Versatilité:** Il peut être appliqué à une large gamme de problèmes de recherche de chemin.

Inconvénients:

- **Arêtes pondérées non négatives:** L'algorithme suppose que les arêtes ont des poids non négatifs.
- **Graphiques cycliques:** Il peut ne pas fonctionner correctement avec des graphes contenant des cycles (boucles).

En résumé, l'algorithme de Dijkstra est un outil puissant et polyvalent pour la recherche de chemin dans les graphes. Sa simplicité, son efficacité et sa versatilité en font un choix populaire dans de nombreux domaines.

Le Pathfinding

Imaginez un robot explorant une forêt dense. Son objectif ? Trouver la source d'un signal mystérieux. Mais pour y parvenir, il doit se frayer un chemin à travers un labyrinthe d'arbres, de buissons et d'obstacles. C'est là qu'intervient le **pathfinding**, ou recherche de chemin.

Qu'est-ce que le pathfinding ?

Le pathfinding est un ensemble de techniques informatiques qui permettent de trouver le chemin le plus optimal entre deux points dans un environnement complexe. Il s'agit d'un problème fondamental dans de nombreux domaines, notamment la robotique, les jeux vidéo, la cartographie et la logistique.

Comment fonctionne le pathfinding ?

L'environnement est généralement représenté par un **graphe**, c'est-à-dire un réseau de points (appelés **nœuds**) reliés par des lignes (appelées **arêtes**). Chaque arête peut avoir un **coût** associé, qui représente la difficulté de la traverser (par exemple, un terrain accidenté aura un coût plus élevé qu'un chemin plat).

L'objectif du pathfinding est de trouver le **chemin le plus court**, ou le **chemin le plus optimal**, entre deux nœuds spécifiques dans ce graphe. Il existe plusieurs algorithmes de pathfinding différents, chacun avec ses propres avantages et inconvénients.

Algorithmes de pathfinding courants :

- **Algorithme de Dijkstra:** Trouve le chemin le plus court avec des coûts non négatifs.
- **A* (A star)*:** Utilise une heuristique pour estimer la distance restante jusqu'au but, ce qui le rend souvent plus rapide que Dijkstra.
- **Recherche en profondeur:** Explore toutes les branches possibles du graphe jusqu'à trouver le but.
- **Recherche en largeur:** Explore les nœuds niveau par niveau, en partant du nœud de départ.

Le choix de l'algorithme dépend de plusieurs facteurs, tels que la taille du graphe, la complexité de l'environnement et les contraintes de temps.

Applications du pathfinding :

- **Robotique:** Pour guider les robots autonomes dans leur environnement.
- **Jeux vidéo:** Pour créer des personnages non joueurs (PNJ) qui se déplacent de manière réaliste.
- **Cartographie:** Pour générer des itinéraires routiers ou des plans d'évacuation.
- **Logistique:** Pour optimiser les livraisons et les flux de marchandises.

Le pathfinding est un domaine vaste et en constante évolution. De nouvelles techniques et de nouveaux algorithmes sont développés en permanence pour améliorer l'efficacité et la précision de la recherche de chemin dans des environnements de plus en plus complexes.

En résumé, le pathfinding est un outil essentiel pour naviguer dans des environnements complexes et trouver le chemin le plus optimal entre deux points. Son utilisation s'étend à de nombreux domaines et continue d'évoluer pour répondre aux besoins croissants d'un monde connecté et automatisé.

Station

La partie "Station" de notre système gère l'analyse de la carte, les ressources et les robots. Une station est initialisée dans le fichier `main.rs` pour analyser la map et effectuer des missions.

Initialisation de la Station

La station est initialisée avec différents paramètres comme la taille de la map, le nombre minimal et maximal de ressources, et le nom de la station. Elle commence toujours en phase d'initialisation.

Analyse de la Carte

La station analyse la carte pour obtenir des données sur les ressources et les robots. La carte est constituée à partir d'un tileset et les cases sont obtenues grâce à une fonction de bruit. Cette analyse est effectuée lorsque la station est en phase d'analyse.

Déplacement des Robots et Vérification des Missions

Les robots sur la carte se déplacent et accomplissent des missions. La station vérifie ces missions et met à jour l'état des robots et des ressources.

Contrôle et Redémarrage de la Simulation

La station peut redémarrer une nouvelle mission en réinitialisant la carte et en augmentant l'époque. Cela permet de lancer de nouvelles analyses et de continuer l'exploration des ressources.

Gestion de la Musique

La station peut jouer de la musique galactique pendant les missions, ce qui ajoute une ambiance immersive à l'expérience.

En résumé, la station est un élément central de notre système, permettant l'analyse et la gestion des ressources et des robots sur la carte. Elle offre des fonctionnalités variées pour initialiser, analyser, contrôler et redémarrer les missions, tout en fournissant une ambiance sonore agréable pour les utilisateurs.

Map

Notre système de biome est géré par un tileset. Un tileset est un fichier contenant des cases (ou tuiles) utilisées pour construire la carte. Il s'agit d'une image unique qui contient toutes les tiles nécessaires pour représenter les différents éléments de la map, ce qui permet de gérer plus efficacement les textures.

Utilisation du Tileset

Nous utilisons le procédé de tileset qui consiste à récupérer les textures à partir d'une seule image contenant toutes les tiles (tuiles ou cases) nécessaires, plutôt que plusieurs images pour chacune des textures. Cela permet de réduire la consommation de mémoire et d'améliorer les performances.

Déplacement sur la Carte

Pour permettre le déplacement avec la souris, ce qui est particulièrement utile lorsque la carte est trop large, nous avons mis en place un système de navigation adapté.

Initialisation des Types de Tuiles

Les types de tuiles sont initialisés en fonction de leur position sur le tileset. Cela permet une attribution précise et ordonnée des différentes textures de la carte, facilitant ainsi la gestion et la mise à jour des éléments graphiques.

Génération du Bruit

Pour ajouter un peu plus de contraste à la carte, nous générons du bruit en fonction de la taille de l'écran. Ce bruit est centré en divisant par 2 et normalisé en divisant par 10. Cela permet de mieux contrôler les variations du bruit et d'obtenir une carte plus uniforme.

Changelog

0.1.0

- Version initiale avec une carte 2D, des robots de base et une station centrale.

0.2.0

- Ajout de la spécialisation des robots pour l'analyse chimique, le forage et l'imagerie.
- Implémentation de la reconnaissance du terrain et de la collecte d'échantillons.
- Amélioration de la communication et du partage d'informations entre les robots.

ADR

ADR-1 : Conception modulaire des robots

- **Motivation** : Permettre des configurations flexibles des robots pour différentes missions.
- **Solution** : Définir des modules spécialisés pour des tâches spécifiques (analyse, forage, imagerie, etc.).
- **Avantages** :
 - Flexibilité et adaptabilité des robots.
 - Facilite la maintenance et l'évolution du code.
- **Inconvénients** :
 - Augmentation de la complexité de la conception.
 - Nécessite une gestion attentive des interfaces entre les modules.

Décision : Implémenter une conception modulaire pour les robots avec des modules spécialisés pour maximiser la flexibilité et l'adaptabilité.

ADR-2 : Stratégie de communication entre les robots

Motivation : Définir une stratégie efficace pour la communication et le partage d'informations entre les robots.

Solutions envisagées :

- **Diffusion** : Les robots diffusent périodiquement leurs découvertes à tous les robots à portée de portée.
- **Communication** : Les robots échangent des informations avec les robots qu'ils rencontrent, propageant les informations à travers le réseau.
- **Routage** : Les robots envoient des messages ciblés à d'autres robots en fonction de leurs besoins spécifiques.

Avantages et inconvénients :

Solution	Avantages	Inconvénients
Diffusion	Simple à implémenter, faible latence pour les mises à jour locales	Inefficace pour les grandes cartes, peut entraîner une surcharge du réseau
Gossip	Robuste aux pannes de robots, échelonnable	Peut entraîner des délais de propagation des informations
Routage	Efficace pour les communications ciblées, réduit la surcharge du réseau	Plus complexe à implémenter, nécessite une connaissance de la topologie du réseau

drive_spreadsheetExporter vers Sheets

Décision : Implémenter une stratégie de communication hybride combinant la diffusion et le routage. La diffusion sera utilisée pour les mises à jour locales et la découverte de

robots, tandis que le routage sera utilisé pour les communications ciblées et le transfert de données volumineuses.

ADR-3 : Modèle de décision pour la création de robots

Motivation : Déterminer comment et quand la station centrale doit créer de nouveaux robots.

Solutions envisagées :

- **Basé sur les ressources** : Créer de nouveaux robots lorsque l'énergie ou les minerais sont suffisants.
- **Basé sur la couverture** : Créer de nouveaux robots pour explorer les zones non cartographiées de la carte.
- **Basé sur les objectifs** : Créer de nouveaux robots en fonction des objectifs de mission spécifiques, tels que la collecte d'échantillons ou la recherche de points d'intérêt scientifiques.

Avantages et inconvénients :

Solution	Avantages	Inconvénients
Basé sur les ressources	Simple à implémenter, garantit que les robots ont les ressources nécessaires pour fonctionner	Peut retarder la création de robots si les ressources sont rares
Basé sur la couverture	Garantit une exploration complète de la carte	Peut créer des robots redondants si la carte est déjà bien explorée
Basé sur les objectifs	Optimise la création de robots pour les objectifs de la mission	Plus complexe à implémenter, nécessite une planification et une coordination précises

Décision : Implémenter un modèle de décision hybride basé sur les ressources et les objectifs. La station centrale créera de nouveaux robots en fonction de la disponibilité des ressources et des objectifs de mission prioritaires.