Simulation feu de Forêt

Simulation feu de forêt

Les paramètres de la simulation sont lus à partir d'un fichier de configuration JSON et sont utilisés pour initialiser la grille de simulation.

Au départ, une propagation initiale est appliquée à partir des positions spécifiées, puis la méthode simuler() est appelée. Cette méthode est conçue pour poursuivre la propagation du feu à chaque étape tant qu'il y a encore des arbres en feu sur la grille.

La méthode propagerFeu() est utilisée pour propager le feu à partir des arbres en feu voisins et pour mettre à jour la grille. Pour déterminer si la simulation est terminée, la méthode estEnFeu() est utilisée. Enfin, à chaque étape, la méthode afficherGrille() est utilisée pour afficher la grille de simulation.

Pour lire le fichier de configuration JSON et initialiser les paramètres de la simulation, le programme utilise la bibliothèque Gson.

Constructeur de la classe Simulation

```
public Simulation(int hauteur, int largeur, double probabilitePropagation,
List<Integer> positionsInitiales) {
this.hauteur = hauteur;
this.largeur = largeur;
this.probabilitePropagation = probabilitePropagation;
this.grille = new boolean[hauteur][largeur];
this.arbreBrule = new boolean[hauteur][largeur];

for(int i = 0; i < positionsInitiales.size(); i += 2) {
int x = (Integer)positionsInitiales.get(i);
int y = (Integer)positionsInitiales.get(i + 1);
this.grille[x][y] = true;
this.arbreBrule[x][y] = true;
}


}
```

"Simulation" prend plusieurs arguments, tels que la hauteur, la largeur, la probabilité de propagation et une liste de positions initiales.

D'abord, je crée deux variables de hauteur et de largeur qui sont initialisées avec les valeurs passées en paramètres. J'ai également une variable de probabilité de propagation qui est initialisée avec une valeur passée en paramètres.

Ensuite, je crée deux tableaux de booléens, l'un pour la grille et l'autre pour les arbres en feu. Ces tableaux ont la même taille hauteur et la largeur passées en paramètres.

Je parcours ensuite les positions initiales à l'aide d'une boucle "for". J'initialise la boucle à zéro et je continue tant que j'ai une paire de valeurs à traiter. J'ajoute 2 à chaque itération de la boucle car chaque paire de valeurs représente une position initiale de l'incendie.

Je récupère ensuite les deux valeurs de chaque paire de valeurs, qui sont les coordonnées x et y de chaque position initiale. J'utilise ensuite ces coordonnées pour accéder à la grille et marquer l'emplacement comme étant en feu dans la grille et dans le tableau d'arbres en feu.

Pour résumer, cette méthode initialise les variables nécessaires pour simuler un incendie dans une grille avec des arbres, en marquant les positions initiales de l'incendie comme étant en feu dans la grille et le tableau d'arbres en feu.

Pourquoi 2 grilles ?

La grille principale "grille[][]" stocke l'état actuel de la forêt. Chaque élément de la grille représente un emplacement dans la forêt et est mis à true lorsqu'un arbre est en feu.

Pour éviter de brûler plusieurs fois les mêmes arbres pendant une étape de simulation, j'utilise la grille auxiliaire "arbreBrule[][]" pour stocker les nouveaux arbres en feu. Lorsqu'un arbre est en feu, il ne brûle pas immédiatement, mais doit attendre l'étape suivante de la simulation pour se propager aux arbres adjacents. Je transfère ensuite les nouveaux arbres en feu de la grille "arbreBrule[][]" à la grille principale "grille[][]" à la fin de chaque étape de simulation.

Ces deux grilles sont nécessaires pour gérer la propagation du feu dans la forêt et éviter de brûler plusieurs fois les mêmes arbres pendant la même étape de simulation.

Méthode main

```
public static void main(String[] args) {
lireConfiguration("config.json");
}
```

méthode principale qui appelle la méthode "lireConfiguration("config.json")".

Méthode lireConfiguratioComment est représentée la grille de simulation dans le code ?n

```
private static void lireConfiguration(String fichier) {
Gson gson = new Gson();

try {
Scanner scanner = new Scanner(new File(fichier));
String json = scanner.useDelimiter("\\Z").next();
scanner.close();
Parametres parametres = (Parametres)gson.fromJson(json, Parametres.class);
Simulation simulation = new Simulation(parametres.hauteur, parametres.longueur,
parametres.probabilitePropagation, parametres.positionsInitiales);
simulation.simuler();
} catch (FileNotFoundException var6) {
System.out.println("Fichier de configuration introuvable : " + fichier);
}


}
```

La méthode "lireConfiguration" est marquée comme privée et statique, ce qui signifie qu'elle ne peut être appelée que dans la classe où elle se trouve et qu'elle ne peut pas être modifiée par d'autres parties du programme.

La méthode commence par créer une instance de la classe Gson, qui est une bibliothèque Java permettant de convertir des objets Java en format JSON et vice versa.

Ensuite, elle ouvre le fichier spécifié en utilisant la classe Scanner de Java. Elle lit le contenu du fichier et le stocke dans une variable appelée "json". Ensuite, elle ferme le scanner pour libérer les ressources.

Ensuite, la méthode utilise la méthode "fromJson" de l'instance Gson pour convertir la chaîne de caractères "json" en un objet de la classe Parametres. Cette classe contient les paramètres de configuration pour la simulation d'incendie tels que la hauteur et la largeur de la grille, la probabilité de propagation du feu, et les positions initiales des arbres en feu.

La méthode crée ensuite une instance de la classe Simulation en utilisant les paramètres lus à partir du fichier de configuration. La méthode appelle la méthode "simuler" de l'objet Simulation pour lancer la simulation.

En cas d'erreur lors de la lecture du fichier, la méthode affiche un message d'erreur indiquant que le fichier n'a pas été trouvé.

Méthode simuler

```
public void simuler() {
int t = 0;
while (estEnFeu()) {
System.out.println("Étape " + t + " :");
afficherGrille();

// Mettre à jour la grille
boolean[][] nouvelleGrille = new boolean[hauteur][largeur];
for (int ligne = 0; ligne < hauteur; ligne++) {
for (int colonne = 0; colonne < largeur; colonne++) {
if (grille[ligne][colonne]) {
propagerFeu(arbreBrule, nouvelleGrille, ligne, colonne);
} else {
// La case n'est pas en feu, elle reste comme elle est
nouvelleGrille[ligne][colonne] = grille[ligne][colonne];
}
}
}
grille = nouvelleGrille;

t++;
}
System.out.println("Terminé après " + t + " étapes.");
}
```

Cette méthode implémente la simulation de la propagation du feu de forêt en mettant à jour la grille à chaque étape jusqu'à ce qu'il n'y ait plus de cellules en feu.

La méthode utilise une boucle "while" qui continue d'exécuter le code tant que la méthode "estEnFeu" retourne vrai. Cette méthode vérifie si au moins un arbre de la grille est en feu.

Dans la boucle "while", le code commence par afficher la grille actuelle avec la méthode "afficherGrille". Ensuite, il met à jour la grille en créant une nouvelle grille vide de même dimension que la grille actuelle.

Ensuite, il examine chaque case de la grille actuelle. Si la case contient un arbre en feu, la méthode "propagerFeu" est appelée pour propager le feu aux arbres voisins dans la nouvelle grille. Si la case ne contient pas d'arbre en feu, elle est simplement copiée de la grille actuelle à la nouvelle grille.

Enfin, la nouvelle grille est assignée à la grille actuelle, le compteur d'étapes "t" est incrémenté, et le processus recommence jusqu'à ce que tous les arbres soient éteints.

Finalement, la méthode affiche un message indiquant le nombre d'étapes nécessaires pour éteindre le feu.

Méthode propagerFeu

```
private void propagerFeu(boolean[][] arbreBrule, boolean[][] nouvelleGrille, int
ligne, int colonne) {
// La case est en feu, elle s'éteint
nouvelleGrille[ligne][colonne] = false; // Sert à stocker les arbres qui ne sont pas
en feu
arbreBrule[ligne][colonne] = true; // Sert à stocker les arbres qui sont en feu
// Propager le feu aux cases adjacentes avec une certaine probabilité
if (ligne > 0 && !arbreBrule[ligne -1][colonne] && Math.random() <
probabilitePropagation) { // Si la ligne > 0 cela signifie que la case a une case
adjacente en haut qui pourrait potentiellement propager le feu, si cette case n'est
pas déjà en feu et que la probabilité de propagation est respectée
nouvelleGrille[ligne -1][colonne] = true; // alors la casedu haut est mise en feu et
la grille mise à jour.
System.out.println("Feu propagé en (" + (ligne -1) + ", " + colonne + ")");
}
if (colonne > 0 && !arbreBrule[ligne][colonne -1] && Math.random() <
probabilitePropagation) { // Case à gauche
nouvelleGrille[ligne][colonne -1] = true;
System.out.println("Feu propagé en (" + ligne + ", " + (colonne -1) + ")");
}
if (ligne < hauteur-1 && !arbreBrule[ligne +1][colonne] && Math.random() <
probabilitePropagation) { // Case en bas
nouvelleGrille[ligne +1][colonne] = true;
System.out.println("Feu propagé en (" + (ligne +1) + ", " + colonne + ")");
}
if (colonne < largeur -1 && !arbreBrule[ligne][colonne +1] && Math.random() <
probabilitePropagation) { // Case à droite
nouvelleGrille[ligne][colonne +1] = true;
System.out.println("Feu propagé en (" + ligne + ", " + (colonne +1) + ")");
}
}
```

Ce code implémente la méthode "propagerFeu" qui est appelée pour propager le feu à partir d'un arbre enflammé à ses voisins. La méthode prend en entrée deux tableaux de booléens, "arbreBrule" et "nouvelleGrille", ainsi que les coordonnées de l'arbre enflammé.

La méthode modifie les tableaux "arbreBrule" et "nouvelleGrille" pour mettre à jour l'état des arbres brûlés et ceux qui viennent de prendre feu. Elle propage le feu à chaque arbre non enflammé qui se trouve à proximité de l'arbre enflammé, avec une probabilité donnée par le champ "probabilitePropagation".

La méthode vérifie d'abord que l'arbre enflammé n'est pas lui-même déjà en feu dans la grille actuelle. Ensuite, elle vérifie si chaque voisin est non enflammé et que la probabilité de propagation est vérifiée. Si c'est le cas, elle met à jour la grille et affiche un message pour indiquer que le feu a été propagé à cette position.

En résumé, cette méthode permet de propager le feu à partir d'un arbre enflammé à ses voisins, en fonction d'une probabilité donnée, et met à jour les tableaux "arbreBrule" et "nouvelleGrille" pour refléter l'état des arbres en feu.

```
private boolean estEnFeu() {
for (int ligne = 0; ligne < hauteur; ligne++) {
for (int colonne = 0; colonne < largeur; colonne++) {
if (grille[ligne][colonne]) {
return true;
}
}
}
return false;
}
```

Ce code est une méthode qui vérifie si un arbre est en feu dans la grille de la forêt. Elle parcourt chaque élément de la grille et vérifie si l'élément correspondant est true, ce qui indique qu'un arbre est en feu. Si un arbre en feu est trouvé, la méthode retourne true. Sinon, si aucun arbre en feu n'est trouvé après avoir parcouru tous les éléments de la grille, la méthode retourne false.

```
private void afficherGrille() {
for (int hauteurIndex = 0; hauteurIndex < hauteur; hauteurIndex++) {
for (int largeurIndex = 0; largeurIndex < largeur; largeurIndex++) {
if (grille[hauteurIndex][largeurIndex]) {
System.out.print("X "); // X représente un arbre en feu
} else if (arbreBrule[hauteurIndex][largeurIndex]) {
System.out.print(". "); // . représente un arbre brulé
} else {
System.out.print("O "); // O représente un arbre sain qui n'est pas en feu
}
}
System.out.println();
}
System.out.println();
}
}
```

La méthode "afficherGrille" affiche la grille de simulation dans la console. Elle parcourt la grille de simulation en utilisant deux boucles "for", une pour la hauteur et une pour la largeur.
Pour chaque case de la grille, la méthode vérifie si l'arbre est en feu, brûlé ou sain en utilisant les tableaux de booléens "grille" et "arbreBrule". En fonction de l'état de l'arbre, un symbole est affiché dans la console: "X" pour un arbre en feu, "." pour un arbre brûlé, et "O" pour un arbre sain. Après avoir parcouru toute la grille, la méthode affiche une ligne vide pour créer un espace entre les affichages successifs de la grille de simulation

Pourquoi choisir d'utiliser la bibliothèque Gson pour lire un fichier JSON ?

La bibliothèque Gson est une bibliothèque Java open-source développée par Google pour convertir des objets Java en format JSON (et vice versa). Elle est largement utilisée dans l'écosystème Java pour faciliter la manipulation de données JSON.

L'utilisation de Gson pour lire un fichier JSON présente plusieurs avantages. Tout d'abord, elle permet de simplifier le code en évitant la nécessité d'écrire des méthodes de parsing JSON manuelles. De plus, la bibliothèque Gson est bien documentée, facile à utiliser et possède de nombreuses fonctionnalités qui permettent de travailler avec des fichiers JSON de manière

efficace. Enfin, la bibliothèque est hautement configurable, ce qui signifie que vous pouvez adapter son comportement à vos besoins spécifiques.

En somme, l'utilisation de la bibliothèque Gson pour lire un fichier JSON permet de gagner du temps et d'éviter les erreurs de parsing, tout en offrant une grande flexibilité dans la manipulation des données JSON.

Quelle est la complexité temporelle de la méthode "simuler" ? Pourquoi ?

La méthode "simuler" permet de simuler un incendie dans une forêt. Pour cela, elle parcourt une grille qui représente la forêt et qui contient des arbres qui peuvent prendre feu. À chaque étape, la méthode vérifie si certains arbres sont en feu et les propage aux arbres voisins. Elle continue ainsi jusqu'à ce qu'il n'y ait plus d'arbres en feu.

La complexité temporelle de la méthode "simuler" correspond au temps que prend la méthode pour parcourir tous les arbres de la grille et propager le feu. Cette complexité dépend du nombre d'étapes nécessaires pour éteindre tous les arbres en feu et de la taille de la grille. Plus la grille est grande et plus il y a d'étapes nécessaires, plus la méthode prendra de temps.

Par exemple, si la grille a 100 cases et qu'il faut 10 étapes pour éteindre tous les arbres en feu, alors la méthode "simuler" prendra 1000 unités de temps. Si la grille est plus grande ou s'il faut plus d'étapes, alors la méthode prendra plus de temps.

Comment améliorer ce code pour qu'il soit plus efficace ou plus facile à maintenir ?

Pour rendre mon code plus facile à maintenir, je pourrais ajouter des commentaires pour expliquer les parties les plus complexes du code ou les choix de conception.

Dans la méthode lireConfiguration, je pourrais peut-être ajouter une vérification des paramètres lus à partir du fichier JSON (par exemple, vérifier que la hauteur et la largeur sont positives, que la probabilité de propagation est entre 0 et 1, etc.) pour éviter les erreurs potentielles pendant l'exécution.

Dans la boucle for qui initialise les grilles avec les positions de départ des incendies, il serait peut-être plus clair d'utiliser un pas de 2 plutôt que d'ajouter 2 à l'index à chaque itération.

Pour rendre mon code plus efficace, il serait possible d'optimiser la méthode propagerFeu en évitant les itérations inutiles sur les cellules qui ne sont pas adjacentes à une cellule en feu. Par exemple, je pourrais ajouter des conditions pour vérifier si la cellule à droite, à gauche, en haut ou en bas est en feu avant de la considérer pour la propagation.

Pour améliorer la lisibilité de mon code, je pourrais renommer les variables grille et arbreBrule pour les rendre plus claires, par exemple grillePrincipale et grilleBrulee.

Il pourrait être utile de déplacer la définition de la variable t dans la méthode simuler() pour éviter qu'elle ne soit accessible depuis d'autres parties du code et pour la limiter à son utilisation dans la boucle while.

Enfin, pour éviter les problèmes potentiels liés à la concurrence dans un environnement multithreadé, je pourrais marquer les méthodes simuler et propagerFeu comme synchronisées.

*Memo: Lorsqu'une méthode est déclarée comme synchronisée, seule une seule instance de la méthode peut être exécutée à la fois, peu importe le nombre de threads qui tentent d'y accéder. Si un thread tente d'exécuter une méthode synchronisée alors qu'une autre instance de la même méthode est déjà en cours d'exécution, le thread sera mis en attente jusqu'à ce que la première instance de la méthode soit terminée. Cela garantit que les threads ne se bousculent pas pour accéder à la même méthode et évite les conflits de données potentiels.

```
public synchronized void maMethode() { // Le code à exécuter en mode synchronisé }
```

Quelles sont les principales limitations ou hypothèses de cette simulation ?

Ce code comporte plusieurs limitations et hypothèses, telles que :

La probabilité de propagation du feu est fixe pour chaque arbre et ne dépend pas des conditions environnementales telles que la vitesse et la direction du vent, l'humidité du sol, etc.

Les arbres sont considérés comme des éléments discrets et identiques, c'est-à-dire qu'il n'y a pas de différenciation entre les espèces d'arbres, leur taille ou leur densité dans la forêt.

La simulation ne prend pas en compte les activités humaines telles que l'exploitation forestière, la construction de routes et de bâtiments, la chasse, etc., qui peuvent affecter la propagation du feu.

Les mesures de prévention et de lutte contre les incendies, telles que la construction de pare-feu, l'utilisation d'hélicoptères pour larguer de l'eau, l'évacuation des personnes, etc., ne sont pas incluses dans la simulation.

Les incendies ne sont pas affectés par les changements saisonniers, tels que les précipitations, les températures et la couverture de neige.

Les arbres en feu ne peuvent pas se rétablir ou se régénérer après la fin de l'incendie. La simulation considère simplement que les arbres qui ont été brûlés sont éliminés de la forêt.

La simulation suppose que la forêt est plate et ne prend pas en compte les différences d'élévation qui peuvent affecter la propagation du feu.

Les arbres en bordure de la grille sont considérés comme étant en dehors de la forêt, de sorte qu'ils ne peuvent pas prendre feu ni propager le feu à l'extérieur de la grille.

Comment garantir que cette simulation est réaliste ou précise ?

Tenir compte de toutes les variables qui peuvent affecter le comportement de la simulation, telles que le vent, l'humidité, la topographie et la densité de la forêt, et de les prendre en compte dans les calculs pour obtenir des résultats précis. Après analyse des résultats de la simulation pour voir s'ils sont conformes aux attentes et aux prévisions. Faire des ajustements en conséquence si néccessaire.