Rapport du projet de programmation orientée objet (POO)

Titre du projet :

Développement d'un Algorithme de Pathfinding (Dijkstra) pour un Robot Naviguant en Java

---

Réalisé par :

-Boussetta Karima **- Ben Salem Noussaiba**

IIA2/1

Encadrante :

Amira Dkhil Jemel , Maître-assistante

Année universitaire : 2024/2025

Résumé du projet :

Ce projet a pour objectif de mettre en œuvre un algorithme de pathfinding (recherche de chemin) en utilisant le Dijkstra Algorithm dans un contexte de robot naviguant dans une grille. Le projet a été réalisé en Java en appliquant les concepts fondamentaux de la Programmation Orientée Objet (POO), tels que l'héritage, l'encapsulation, l'abstraction, les interfaces et les classes abstraites.

Le programme simule le déplacement d'un robot dans une grille 2D, où certains espaces peuvent être occupés par des obstacles. L'objectif est de trouver le chemin le plus court entre un point de départ et une destination donnée en utilisant l'algorithme de Dijkstra. Le projet inclut également une gestion des erreurs pour traiter les situations où aucun chemin n'est possible, ainsi qu'une visualisation du déplacement du robot sur la grille.

I/ Les Classes :

1) Node : Représente un nœud (case de la grille) avec coordonnées et coût.

- **Attributs :**
  - `x, y` : Coordonnées du nœud dans la grille.
  - `cost` : Coût pour atteindre ce nœud depuis le départ.
  - `parent` : Nœud parent pour reconstruire le chemin.
- **Méthodes :**
  - **Accesseurs/Mutateurs** (`getX, setX`, etc.) : Accès et modification des attributs.
  - `equals` et `hashCode` : Comparer les nœuds pour détecter les doublons.
  - **Constructeur** : Initialise un nœud avec un coût infini et pas de parent par défaut.

2) Graph : Modélise la grille avec ses obstacles et les connexions possibles.

- **Attributs :**
  - `grid` : Tableau 2D représentant la grille (0 = libre, 1 = obstacle).
- **Méthodes :**
  - `getGrid()` : Retourne la grille.
  - `getNeighbors(Node node)` : Retourne les voisins accessibles d'un nœud (haut, bas, gauche, droite).

3) PathfindingAlgorithm (abstraite) : Classe abstraite pour les algorithmes de recherche de chemin.

- **Méthode abstraite :**
  - `findShortestPath(Node start, Node destination, Graph graph)` : Définit la recherche de chemin.

4) Dijkstra (implémente PathfindingAlgorithm) :Implémente l'algorithme de recherche de chemin

- **Attributs :**
  - Aucun (l'algorithme travaille avec les paramètres).
- **Méthodes :**
  - `findShortestPath(Node start, Node destination, Graph graph)` :
  - Utilise une file de priorité pour explorer les nœuds accessibles.
  - Met à jour les coûts et les parents des voisins.
  - Reconstruit et retourne le chemin si la destination est atteinte.
  - Lève une exception (`PathfindingException`) si aucun chemin n'est trouvé.

5) Robot (implémente Moveble et Renderable) : Contrôle le robot et effectue les mouvements en utilisant Dijkstra.

- **Attributs**
  - `position` : Position actuelle du robot (type `Node`).
  - `pathfindingAlgorithm` : Instance de l'algorithme utilisé (ici, Dijkstra).
  - `graph` : Grille dans laquelle le robot se déplace.
- **Méthodes :**
  - `move(Node nextPosition)` : Déplace le robot vers une position donnée.
  - `navigate(Node destination)` :
    - Trouve le chemin avec `findShortestPath.`
    - Déplace le robot étape par étape en affichant chaque mouvement.
  - `render()` : Affiche la grille avec la position actuelle du robot (`R` pour robot, `#` pour obstacles, `.` pour cases libres).

6) PathfindingException :

Gère les exceptions liées à l'impossibilité de trouver un chemin.

- **Attributs :**
  - Hérite des attributs de `Exception.`
- **Constructeur :**
  - Reçoit un message personnalisé

7) Main : Point d'entrée du programme (interface utilisateur et navigation).

- **Rôle :**
  - Configure la grille avec des obstacles.
  - Récupère les coordonnées de destination de l'utilisateur.
  - Initialise un robot, une grille et l'algorithme Dijkstra.
  - Appelle `navigate` pour trouver et afficher le chemin.

II/ Les Interfaces :

1) Moveble : Interface définissant les méthodes de mouvement.

- **Méthodes :**
  - `move()` : Méthode pour déplacer l'entité.
  - `move(Node nextPosition)` : Déplace vers une position donnée.

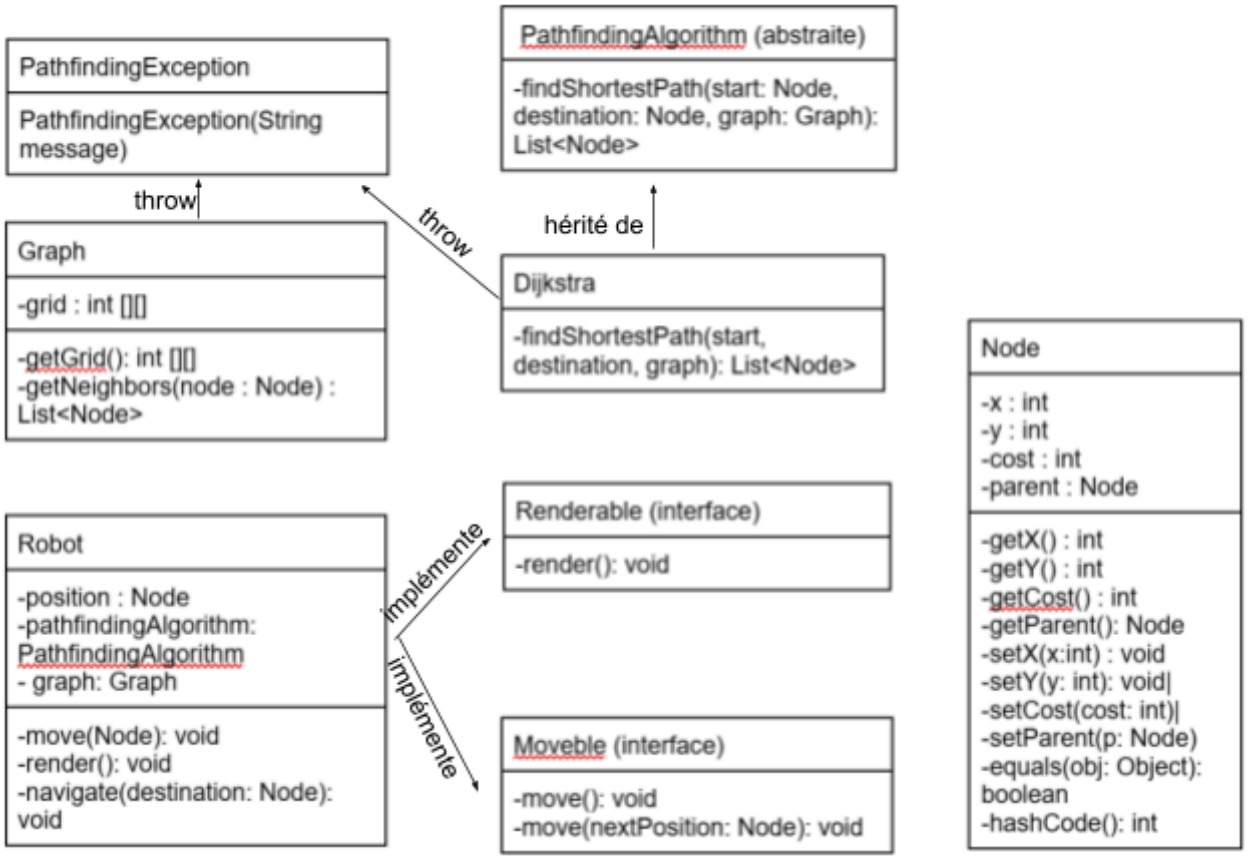
2) Renderable : Interface définissant les méthodes d'affichage.

- **Méthode :**
  - `render()` : Affiche l'état actuel de l'entité dans la grille.

III/ Fonctionnement Général :

1. **Initialisation** :
    - L'utilisateur définit une **destination valide** dans une grille 5x5 contenant des obstacles.
    - Le robot démarre depuis `(0, 0)`.
2. **Algorithme Dijkstra** :
    - Utilise une **file de priorité** pour explorer les nœuds avec le plus faible coût.
    - Met à jour les voisins en enregistrant leur coût et leur parent.
3. **Affichage** :
    - La grille est affichée après chaque déplacement du robot.
    - Le robot est représenté par ***R***, les obstacles par **#**, et les cases libres par **.**.
4. **Navigation** :
    - Le robot suit le chemin trouvé, étape par étape, jusqu'à atteindre la destination.

IV/ Diagramme de Classe : Voici un aperçu des relations entre les classes :



V/ Points Forts :

- **Modularité** : Le code utilise des classes et interfaces bien structurées.
- **Flexibilité** : L'algorithme de pathfinding peut être remplacé par un autre (par exemple, A*).
- **Interactivité** : L'utilisateur choisit la destination.
- **Affichage dynamique** : La grille est mise à jour en temps réel pour suivre les déplacements du robot.

VI/ Conclusion :

Ce projet met en œuvre un robot capable de naviguer dans une grille en évitant les obstacles grâce à l'algorithme de Dijkstra. Le design est modulaire et extensible, facilitant l'intégration d'autres algorithmes de recherche de chemin ou l'amélioration de l'affichage.