**Documentation des Fonctions de l'Application de Triangulation de Delaunay avec Obstacles et Pathfinding**

**DelaunayTriangulationApp (Classe Principale)**

**Méthodes :**

**public DelaunayTriangulationApp()**

* Constructeur de la fenêtre principale.
* Initialise les composants de l'interface graphique (boutons, panneaux, événements).
* Ajoute la gestion des clics pour définir les points de départ et d’arrivée.

**private void generateTriangulation()**

* Efface l'ancien maillage et les points générés.
* Ajoute les **quatre points fixes** de la zone.
* Génère un certain nombre de points **aléatoires** (sauf dans les obstacles).
* Crée la triangulation de Delaunay en appelant DelaunayTriangulation.compute().
* Rafraîchit l’affichage.

**private void addObstacle()**

* Génère un **obstacle aléatoire** (rectangle ou triangle).
* Ajoute ses sommets à la liste des obstacles.
* Rafraîchit l'affichage.

**private void clearTriangulation()**

* Réinitialise complètement l’affichage.
* Supprime tous les **points, triangles et obstacles**.
* Réinitialise les **points de départ et d’arrivée**.
* Rafraîchit l'affichage.

**private void handleRightClick(Point p)**

* Gestion du clic droit pour définir **les points de départ et d’arrivée**.
* Si **aucun point n'est défini**, place **startPoint**.
* Si **seul startPoint est défini**, place **endPoint**.
* Si **les deux sont déjà définis**, **remplace startPoint par endPoint** et met à jour endPoint.
* Vérifie que les points ne sont **pas dans un obstacle**.
* Rafraîchit l'affichage.

**private boolean isInsideObstacle(Point p)**

* Vérifie si un point est **à l'intérieur d'un obstacle**.
* Retourne true si le point est dans un obstacle, false sinon.

**private void runPathfinding()**

* Appelle la méthode Pathfinding.findPath(startPoint, endPoint).
* Vérifie que **startPoint et endPoint sont bien définis**.

**DelaunayTriangulation (Classe de Calcul du Maillage)**

**Méthodes :**

**public static void compute(List<Point> points, List<Triangle> triangles, int width, int height)**

* Effectue la **triangulation de Delaunay** avec les points donnés.
* Assure que **les obstacles ne sont pas traversés**.
* Supprime les triangles **invalides**.
* Stocke le résultat dans la liste triangles

**Obstacle (Classe de Gestion des Obstacles)**

**Méthodes :**

**public Obstacle(Point... points)**

* Crée un obstacle avec **3 ou 4 sommets** (triangle ou rectangle).
* Vérifie que la structure est correcte.

**public List<Point> getVertices()**

* Retourne la liste des **sommets de l’obstacle**.

**public boolean contains(Point p)**

* Vérifie si un point est **à l’intérieur de l’obstacle**.

**public void draw(Graphics2D g2d)**

* Dessine l’obstacle sur le panneau de dessin.

**public boolean intersectsObstacle(Edge e)**

* Vérifie si **une arête coupe un obstacle** (utilisé dans DelaunayTriangulation).

**DrawPanel (Classe de Dessin de l'Interface Graphique)**

**Méthodes :**

**protected void paintComponent(Graphics g)**

* Dessine **le maillage, les obstacles et les points**.
* Affiche **startPoint (en vert) et endPoint (en bleu)**.
* Dessine la **zone de maillage** avec un contour noir.

**Principe de la Triangulation de Delaunay**

La triangulation de Delaunay est une méthode de division d'un ensemble de points en triangles de manière à maximiser l’angle minimum de chaque triangle. Cela garantit que les triangles obtenus sont aussi "équidistants" que possible, évitant les triangles fins et allongés. Cette triangulation est utilisée en modélisation 3D, en maillage géométrique et en interpolation.

**Méthode de Bowyer-Watson**

L’algorithme de Bowyer-Watson est une approche incrémentale pour construire une triangulation de Delaunay. Il fonctionne comme suit :

1. Commence par un super-triangle englobant tous les points.
2. Ajoute les points un par un et supprime les triangles dont le cercle circonscrit contient le nouveau point.
3. Relie les bords extérieurs des triangles supprimés pour former de nouveaux triangles conformes à la triangulation de Delaunay.
4. Supprime les triangles dépendants du super-triangle.

Cette méthode est efficace et largement utilisée pour la génération dynamique de maillages.