



Université
Paris Cité

FireGuardian

Un Simulateur Global de Gestion
des Incendies

Développé par :
Alexandre DIAMANT
et
Yahya HAMDI

Quelques Chiffres Clés sur les Feux de Forêt

01.

Superficie Brûlée (2020)

10,274,679 acres

02.

Nombre de Feux(2006)

96,385

03.

Superficie Brûlée/Feu(2020)

47.3 acres

FireGuardian , C'est quoi ?

Un simulateur avancé de gestion des incendies conçu pour **optimiser les interventions des pompiers** en temps réel. Il permet de :

Gestion Avancée des Pompiers

FireGuardian permet de gérer une **flotte diversifiée** de véhicules pompiers, chacun ayant des caractéristiques spécifiques telles que la vitesse, la capacité du réservoir et la capacité de se déplacer sur différents types de terrains.

Extinction des Incendies

Le simulateur génère des incendies de manière aléatoire sur une carte. Les véhicules sont déployés de manière stratégique pour éteindre les incendies en **tenant compte des variables** telles que l'intensité du feu et le type de terrain.

Optimisation des Trajets

Grâce aux **algorithmes A*** et **des lucioles**, FireGuardian trouve les chemins les plus rapides pour les véhicules, que ce soit pour atteindre les incendies ou pour se rendre aux points de remplissage d'eau.

Génération du Monde

FireGuardian peut être adapté à **différentes régions géographiques**. Il suffit de choisir les régions déjà présentes dans le logiciel ou d'ajouter sa propre région facilement.

Scénarios d'Utilisation

Intervention en Milieu Urbain

- FireGuardian a choisi de deployer le **camion**
- Exploiter la grande **capacité** du réservoir
- Capacité de circuler rapidement sur les routes goudronnées.

En Milieu Montagneux

- FireGuardian a choisi de deployer le **véhicule 4X4**
- Capacité de circuler sur des **terrains difficiles** avec une vitesse modérée

Intervention en Milieu Isolé

- FireGuardian a choisi de deployer **l'avion pompier**
- Capacité de se déplacer **directement** vers le site de l'incendie sans suivre les chemins **existants**

FireGuardian : En Plusieurs Parties, Ça Donne Quoi ?

01.

Génération du Monde

- Création de la carte avec différents types de terrains, des feux et des casernes.
- Génération de multiples cartes basées sur des régions du monde réel.

02.

Recherche du chemin et du véhicule

- Utilisation de l'algorithme A* pour déterminer les trajets les plus courts.
- Sélection du véhicule approprié en fonction du type de terrain et de l'intensité du feu

03.

Optimisation de la Recherche

- Implémentation d'un algorithme génétique pour améliorer les solutions initiales.
- Trouver des stratégies d'intervention optimales dans le cas de **feux multiples**

Chronologie et Repartition des tâches

Oct 23

Développement du 1er projet : MicroMouse

Break du 1er Semestre

Fin

Fev 24

Abandon du développement du projet MicroMouse

Proposition du sujet FireGuardian

Debut

Mars 24

Recherche de ressources pour la generation et la modelisation du monde

Recherche d'articles scientifiques pour la conception de l'algorithme

Debut

Avril 24

Premier prototype de generation du monde terminé

Visualiseur de l'algorithme des lucioles terminé

Debut de l'implementation de la partie algorithmique dans le projet

Mi-Avril

24

Premier prototype de FireGuardian

Découverte de problemes d'optimisation au niveau de l'algorithme

Fin Avril
24

Résolution de problèmes d'optimisation pour les cartes de taille reduite

Création de l'UI

Debut
Mai 24

Amélioration de l'interface graphique

Affichage du resultat de l'algorithme

Mi-Mai
24

Résolution de bugs

Preparation du rendu finale

Les taches de Yahya

- Recherche scientifique pour les algorithmes génétique et l'optimisation discrete.
- Développement et optimisation de l'algorithme des lucioles et A*.
- Création du visualiseur pour l'algorithme des lucioles.

Les taches d'Alexandre

- Développement du visualiseur du Monde et l'interface graphique
- Contribution à l'implémentaion des algorithmes utilisés
- Recherche de ressources pour la génération du monde

Compétences Techniques Nécessaires

Programmation Java et Orienté Objet

Lecture d'articles scientifiques

Design Patterns et UML

Tests unitaires

Choix de Conception

- Division du projet en **deux parties indépendantes** pour permettre un développement parallèle sans interdépendance
- Choix guidé par le temps **limité** de développement restant.

Modulaire

- Répartition en plusieurs classes et paquets pour une meilleure organisation.

**Notre
Architecture
est :**

Evolutive

- Les parties majeures du projet sont facilement modifiables pour permettre des changements ou l'ajout de nouvelles fonctionnalités.

Robuste

- Utilisation de tests unitaires et de validation des algorithmes pour garantir la fiabilité et la stabilité du logiciel même en cas de modifications majeures.

Structure et Paquets du Logiciel

Le paquet backend

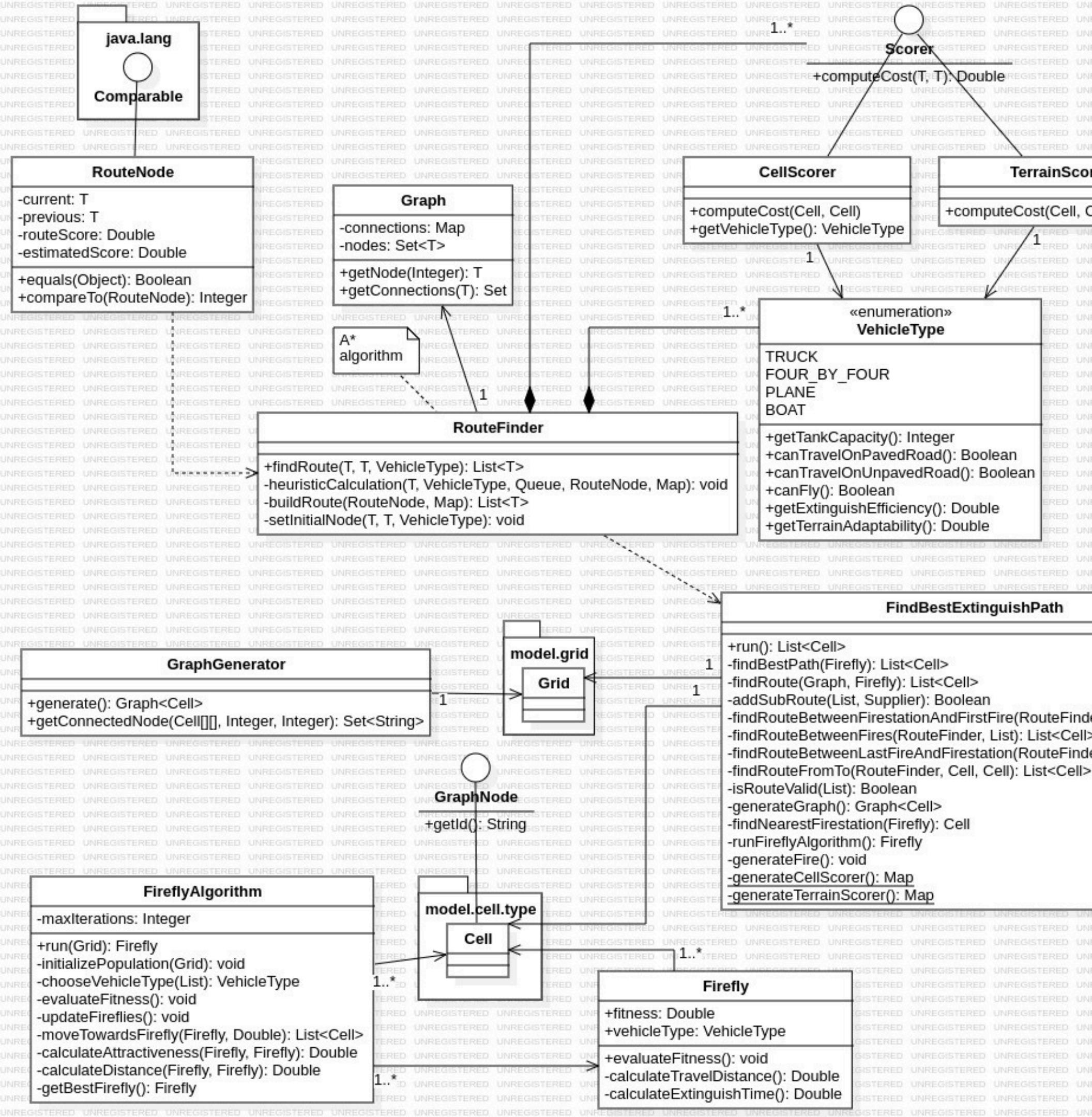
- **Algo**
 - RouteFinder.java
 - FireFly.java
 - Graph.java
 - ...
- **Model**
 - cell
 - Fire
 - grid
 - ...
- **Io**
 - CommandsFormat.java
- **utils**
 - Constants.java
 - ...

Le paquet Frontend

- **button**
- **label**
- **filechooser**
 - ChoiceMapfileChooser.java
- **panel**
- **listener**
 - ScaleFieldlistener.java
 - ...
- **observer**
 - Scaleobserver.java
 - ResultAlgorithmobserver.java
 - ...
- **window**
 - window.java
- **textfield**

UML

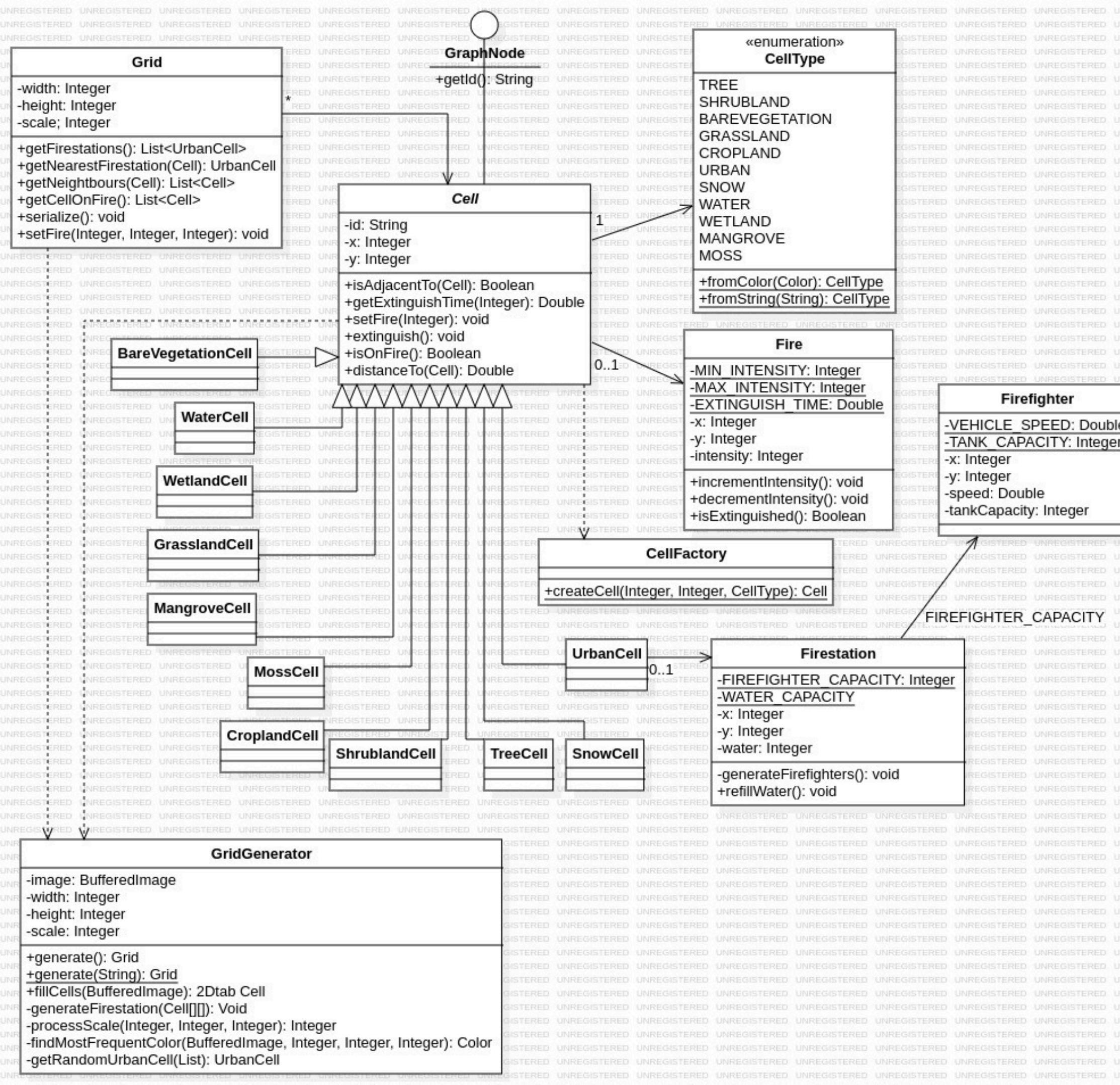
Algorithm



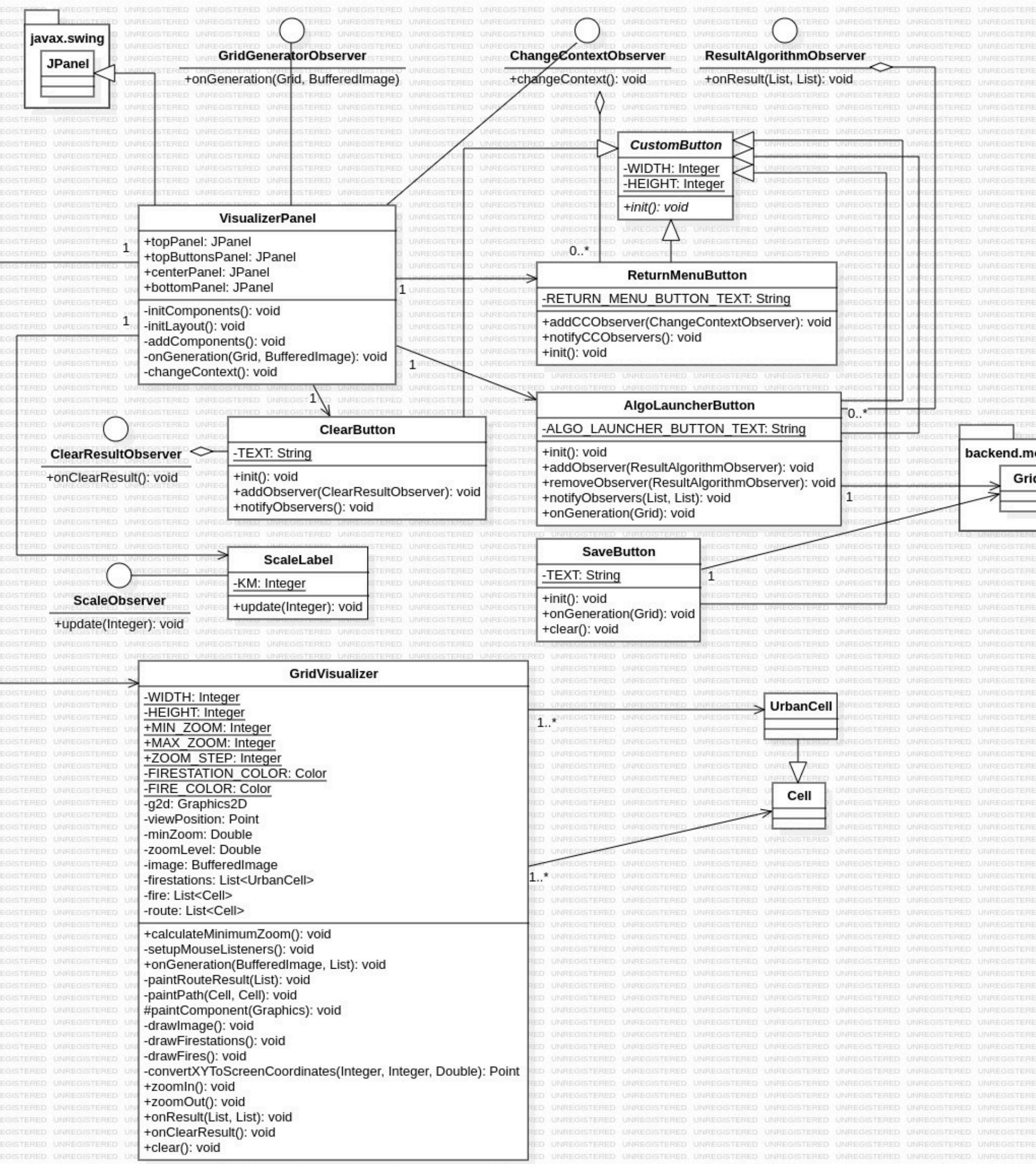
UML

Génération du

monde



UML Visualiseur



Difficultés Techniques et Solutions

Algorithme :

Temps d'exécution exponentiel par rapport à la taille de la carte.

Optimisation en Modifiant l'Heuristique et en Utilisant le Parallélisme

Visualiseur :

Difficulté rencontrée avec la taille des cartes utilisées.

Création d'une image par rapport à la grille au lieu de simplement récupérer la carte.

FireFly.java

```
public class Firefly {  
    private List<Cell> fireSequence;  
    private double fitness;  
    private VehicleType vehicleType;  
  
    public Firefly(List<Cell> fireSequence, VehicleType vehicleType) {  
        this.fireSequence = fireSequence;  
        this.vehicleType = vehicleType;  
    }  
  
    public void evaluateFitness() {  
        if (vehicleType == null) {  
            throw new IllegalStateException("Vehicle type is not set for this firefly.");  
        }  
        double totalTravelDistance = calculateTravelDistance();  
        double totalExtinguishTime = calculateExtinguishTime();  
        double speedFactor = vehicleType.getSpeed();  
        double tankFactor = vehicleType.getTankCapacity();  
        double efficiencyFactor = vehicleType.getExtinguishEfficiency();  
        double terrainAdaptabilityFactor = vehicleType.getTerrainAdaptability();  
  
        //penalite pour les vehicules avec une capacite de reservoir inferieure a 3  
        double extinguishPenalty = (tankFactor < 3) ? 2.0 : 1.0;  
  
        fitness = 1 / ((totalTravelDistance / speedFactor)  
                      + (totalExtinguishTime * extinguishPenalty / efficiencyFactor)  
                      + terrainAdaptabilityFactor + 0.01);  
    }  
}
```

TerrainCosts.java

```
public static Map<VehicleType, Double> calculateTotalCosts(Grid grid, List<Cell> fireSequence)  
{  
    Map<VehicleType, Double> totalCosts = new EnumMap<>(VehicleType.class);  
    for (VehicleType vehicleType : VehicleType.values()) {  
        double totalCost = 0;  
        for (Cell cell : fireSequence) {  
            totalCost += getCost(vehicleType, cell);  
        }  
        totalCosts.put(vehicleType, totalCost);  
    }  
    return totalCosts;  
}  
  
private static void initFourByFourCosts() {  
    Map<Celltype, Double> costs = new EnumMap<>(Celltype.class);  
    costs.put(Celltype.URBAN, 1.0);  
    costs.put(Celltype.GRASSLAND, 1.2);  
    costs.put(Celltype.WATER, Double.POSITIVE_INFINITY);  
    costs.put(Celltype.MANGROVE, 2.0);  
    costs.put(Celltype.TREE, 1.5);  
    costs.put(Celltype.CROPLAND, 1.0);  
    costs.put(Celltype.BAREVEGETATION, 1.2);  
    costs.put(Celltype.SHRUBLAND, 1.3);  
    costs.put(Celltype.SNOW, 2.0);  
    costs.put(Celltype.WETLAND, 1.5);  
    costs.put(Celltype.MOSS, 1.5);  
    vehicleTerrainCosts.put(VehicleType.FOUR_BY_FOUR, costs);  
}
```

FireFlyAlgorithm.java

```
public Firefly run(Grid grid) {
    double bestFitness = Double.NEGATIVE_INFINITY;
    Firefly bestFirefly = null;
    int stableIterations = 0;

    fires = grid.getCellOnFire();
    initializePopulation(grid);

    for (int i = 0; i < maxIterations; i++) {
        evaluateFitness();

        updateFireflies();

        Firefly currentBest = getBestFirefly();
        if (currentBest.getFitness() > bestFitness) {
            bestFitness = currentBest.getFitness();
            bestFirefly = currentBest;
            stableIterations = 0;
        } else {
            stableIterations++;
        }

        if (stableIterations >= MAX_STABLE_ITERATIONS) {
            initializePopulation(grid);
            stableIterations = 0;
        }
    }

    return bestFirefly;
}
```

```
private VehicleType chooseVehicleType(Grid grid, List<Cell> fireSequence) {
    double minCost = Double.POSITIVE_INFINITY;
    VehicleType chosenType = null;

    for (VehicleType vehicleType : VehicleType.values()) {
        double totalTravelDistance = 0;
        double totalExtinguishTime = 0;
        double speedFactor = vehicleType.getSpeed();
        double tankFactor = vehicleType.getTankCapacity();
        double efficiencyFactor = vehicleType.getExtinguishEfficiency();
        double terrainAdaptabilityFactor = vehicleType.getTerrainAdaptability();
        double terrainCost = 0;
        double extinguishPenalty = (tankFactor < 3) ? 2.0 : 1.0;

        for (Cell cell : fireSequence) {
            terrainCost += TerrainCosts.getCost(vehicleType, cell);
            totalExtinguishTime += cell.getExtinguishTime(vehicleType.getTankCapacity());
        }

        double totalCost = terrainCost + (totalTravelDistance / speedFactor)
            + (totalExtinguishTime * extinguishPenalty / efficiencyFactor)
            + terrainAdaptabilityFactor;

        if (totalCost < minCost) {
            minCost = totalCost;
            chosenType = vehicleType;
        }
    }

    return chosenType;
}
```

```
private void updateFireflies() {
    ConcurrentMap<Firefly, Firefly> newFireflySequences = new ConcurrentHashMap<>();
    population.parallelStream().forEach(firefly -> {
        population.parallelStream().forEach(other -> {
            if (firefly == other) return;
            double attractiveness = calculateAttractiveness(firefly, other);
            if (other.getFitness() > firefly.getFitness()) {
                List<Cell> newSequence = moveTowardsFirefly(firefly, attractiveness);
                newFireflySequences.put(firefly, new Firefly(newSequence,
                    firefly.getVehicleType()));
            }
        });
    });

    newFireflySequences.forEach((oldFirefly, newFirefly) -> {
        oldFirefly.setSequence(newFirefly.getFireSequence());
        oldFirefly.setFitness(newFirefly.getFitness());
    });
}

private List<Cell> moveTowardsFirefly(Firefly firefly, double attractiveness) {
    List<Cell> newSequence = new ArrayList<>(firefly.getFireSequence());
    if (attractiveness > ThreadLocalRandom.current().nextDouble()) {
        int i1 = ThreadLocalRandom.current().nextInt(newSequence.size());
        int i2 = ThreadLocalRandom.current().nextInt(newSequence.size());
        Collections.swap(newSequence, i1, i2);
    }
    return newSequence;
}
```

RouteFinder.java

```
public List<T> findRoute(T from, T to, VehicleType vehicleType) {
    Queue<RouteNode<T>> openSet = new PriorityQueue<>(RouteNode::compareTo);
    Map<T, RouteNode<T>> allNodes = new HashMap<>();

    setInitialNode(from, to, vehicleType, openSet, allNodes);

    while (!openSet.isEmpty()) {
        RouteNode<T> next = openSet.poll();
        if (next.getCurrent().equals(to)) {
            return buildRoute(next, allNodes);
        }

        heuristicCalculation(to, vehicleType, openSet, next, allNodes);
    }

    return Collections.emptyList();
}
```

```
private void heuristicCalculation(T to, VehicleType vehicleType, Queue<RouteNode<T>> openSet,
RouteNode<T> next, Map<T, RouteNode<T>> allNodes) {
    graph.getConnections(next.getCurrent()).forEach(connection -> {
        RouteNode<T> nextNode = allNodes.getOrDefault(connection, new RouteNode<>(connection));
        allNodes.put(connection, nextNode);

        Scorer<T> cellScorer = cellScorers.get(vehicleType);
        Scorer<T> terrainScorer = terrainScorers.get(vehicleType);

        double newScore = next.getRouteScore() + cellScorer.computeCost(next.getCurrent(),
connection);
        if (newScore < nextNode.getRouteScore()) {
            nextNode.setPrevious(next.getCurrent());
            nextNode.setRouteScore(newScore);
            nextNode.setEstimatedScore(newScore + terrainScorer.computeCost(connection, to));
            openSet.add(nextNode);
        }
    });
}

private List<T> buildRoute(RouteNode<T> next, Map<T, RouteNode<T>> allNodes) {
    List<T> route = new ArrayList<>();
    RouteNode<T> current = next;
    do {
        route.add(0, current.getCurrent());
        current = allNodes.get(current.getPrevious());
    } while (current != null);

    return route;
}
```

Scénarios Complexes

- Impact des conditions météorologiques sur la propagation des incendies
- Prise en compte des obstacles naturels tels que les rivières, les lacs, les falaises...
- Feux multiples se propageant **simultanément**, nécessitant une coordination complexe et des ressources diversifiées.



Collaboration en Temps Réel

- Ajout d'une fonctionnalité de collaboration en temps réel permettant aux équipes de pompiers de différentes régions de travailler ensemble et de coordonner leurs efforts
- Déploiement de systèmes de communication intégrés pour permettre une coordination fluide et efficace entre les différentes équipes de secours

FireGuardian 2.0

Intégration d'une IA

- **Prédire** l'évolution des incendies en fonction des conditions météorologiques et du type de terrain.
- Utilisation de l'apprentissage automatique pour améliorer continuellement les stratégies d'intervention basées sur les données historiques et les résultats des interventions passées



Interface Utilisateur Améliorée

- Personnalisation des interfaces pour s'adapter aux besoins spécifiques de chaque service de pompiers.
- Interface utilisateur encore plus intuitive et interactive

If we had a Time Machine

- **Étude Approfondie du Sujet**

- Mener une étude approfondie dès le début pour s'assurer que le sujet choisi est le bon

- **Étude de Faisabilité**

- Réaliser une étude de faisabilité par rapport au **temps de développement** restant pour bien déterminer les objectifs à atteindre.
 - Établir un planning réaliste et des jalons clairs pour gérer efficacement le temps



Merci !