

# RAPPORT

## Introduction

Le projet vise à développer un proto-simulateur d'automate cellulaire mettant en scène des pompiers, des feux, des nuages, des pompiers motorisés, des montagnes et des routes sur une grille. L'automate cellulaire simule le comportement de ces entités, où à chaque tour, les pompiers agissent pour éteindre les feux, les feux se propagent, les nuages éteignent les feux, et des éléments spécifiques comme les pompiers motorisés, les montagnes, et les routes influent sur le déplacement et l'interaction des entités.

## Objectifs

Le principal objectif du projet est de créer un simulateur respectant les principes SOLID, une série de principes de conception logicielle favorisant la flexibilité, la maintenabilité et la compréhension du code. Le projet initial fournit un point de départ, mais le code ne respecte pas les principes SOLID, ce qui nécessite une refonte pour assurer la qualité du logiciel.

## Architecture du Code

La première tâche consiste à établir une architecture de code qui respecte les principes SOLID. Cela implique de restructurer le code pour garantir que chaque classe a une responsabilité unique, que les dépendances sont inversées, et que les extensions sont favorisées plutôt que les modifications.

L'utilisation de patrons de conception peut être justifiée pour améliorer la conception. Par exemple, l'application du patron de conception "Stratégie" pourrait rendre le comportement des entités (pompiers, feux, nuages) plus flexible, permettant des variations sans altérer la structure de base.

## Fonctionnalités Étendues

La tâche suivante consiste à étendre le modèle avec plusieurs fonctionnalités supplémentaires :

1. **Nuages Mouvants** : Ajout de nuages se déplaçant aléatoirement et capable d'éteindre les feux. Cela nécessitera une gestion intelligente des déplacements des nuages.
2. **Pompiers Motorisés** : Introduction de pompiers motorisés capables de se déplacer sur une distance plus grande (deux cases). Cela impliquera des ajustements dans la logique de déplacement des pompiers.
3. **Montagnes** : Ajout de cases montagnes qui ne sont pas franchissables par le feu ni par les pompiers. Cela nécessitera une adaptation des mécanismes de déplacement.
4. **Routes** : Intégration de cases routes, accessibles uniquement par les pompiers. Les routes pourraient offrir un chemin plus rapide pour les pompiers.

## Implémentation

Dans cette implémentation, plusieurs ajustements ont été apportés pour améliorer la structure et la maintenabilité du code du simulateur d'automate cellulaire. Tout d'abord, une interface **Model** a été introduite pour gérer les éléments du tableau, suivant le principe d'abstraction.

Ensuite, les classes **Fires** et **FireFighters** ont été créées pour séparer la gestion des feux et des pompiers de la classe **FireFighterBoard**. Cette refonte respecte le principe de responsabilité unique (SRP), assurant que chaque classe a une seule raison de changer, et ouvre la voie à des extensions futures sans modification de code existant (principe OCP).

Le package **View** a également été restructuré en remplaçant l'enum **ViewElement** par une interface éponyme, implémentée par des classes spécifiques à chaque élément du tableau. Cela améliore la flexibilité de la vue en permettant l'ajout facile de nouveaux éléments.

La logique des nuages (Clouds) et des pompiers motorisés (MotorizedFireFighters) a été intégrée dans le package **model.ModelElement**, unifiant la gestion des différents éléments du simulateur.

Les obstacles, représentés par les montagnes et les routes, ont été traités dans le package **model.Obstacles**, introduisant une hiérarchie de classes abstraites (**Obstacles**) pour éviter toute duplication de code et promouvoir la réutilisation.

En résumé, cette réorganisation du code respecte les principes SOLID, rend le code plus modulaire, et facilite l'extension et la maintenance du simulateur d'automate cellulaire.

## Gestion du Projet

Le projet est hébergé sur un dépôt Git, offrant un suivi du code et des fonctionnalités implémentées. Les commits réguliers assurent la traçabilité du travail effectué par les membres du projet. L'utilisation de branches peut faciliter le travail collaboratif et la gestion des différentes fonctionnalités.

## Diagramme de classe

Dans le cadre d'une optimisation de la visibilité et de la lisibilité du présent rapport, le diagramme de classe, détaillant la structure et les relations entre les différentes composantes du simulateur d'automate cellulaire, sera annexé en tant que pièce jointe. Cette démarche vise à offrir une représentation visuelle claire et concise du modèle logiciel élaboré, permettant une compréhension approfondie des interactions entre les classes, interfaces, et autres entités du système. La consultation du diagramme en annexe complètera de manière judicieuse les informations présentées dans le texte, contribuant ainsi à une documentation exhaustive du projet.

## Conclusion

Ce projet de proto-simulateur d'automate cellulaire a transcendé la simple implémentation technique pour embrasser des principes fondamentaux de conception logicielle et une gestion de projet efficace. En respectant rigoureusement les principes SOLID, nous avons métamorphosé le code initial en une base plus robuste, extensible et facilement compréhensible.

L'ensemble de ces ajustements a non seulement produit un code plus propre et structuré, mais a également renforcé notre compréhension des principes SOLID et de leur application pratique dans des projets logiciels complexes. Les engagements réguliers via Git ont démontré l'importance de la traçabilité du travail, favorisant une collaboration efficace entre les membres de l'équipe.

En somme, ce projet va au-delà de la simple programmation, il représente une immersion dans les meilleures pratiques de développement logiciel, allant de la conception à la gestion

de projet. Les leçons apprises enrichiront notre approche future du développement logiciel, faisant de cette expérience un jalon significatif dans notre parcours professionnel.