

AIX-MARSEILLE UNIVERSITE
FACULTE DES SCIENCES - CENTRE DE TELE-ENSEIGNEMENT SCIENCE
LICENCE 3 MATHEMATIQUES INFORMATIQUE

**Simulation d'un feu de forêt à l'aide des automates
cellulaires**
Projet Mathematiques-Informatique

Gladys DJOUFACK TADONKA
Maeva DHAYNAUT
Amine HADDOU
Nassim amar ROUAG
Franck SOULON

Encadrant : **Omar BOUCELMA**
Année académique : **2022/2023**

Mai 2023

Table des matières

Table des figures	I
Liste des tableaux	II
Introduction	1
1 Revue de la littérature	2
2 Quelques concepts théoriques	5
2.1 Automate cellulaire	5
2.2 Diagramme de <i>Gantt</i>	5
2.3 UML	6
2.4 Git	7
2.5 Java	7
3 Cas d'étude : Simulation graphique d'un feu de forêt avec le langage Java	8
3.1 Modélisation mathématique du problème	8
3.1.1 Impact du voisinage	9
3.1.2 Impact de la direction du vent	10
3.1.3 Impact du climat	10
3.1.4 Impact de l'humidité du sol	11
3.1.5 Type de végétation	12
3.2 Gestion de projet	12
3.2.1 Diagramme de Gantt	12
3.2.2 Trello	13
3.2.3 Git	14
3.2.4 Méthodologie Agile	14
3.3 Modélisation informatique du problème	15
3.3.1 Construction de la version initiale du diagramme de classes	16
3.3.2 Amélioration du diagramme	17
3.3.3 Description du diagramme de classes	18
3.4 Design de l'interface graphique	20
3.5 Mise en œuvre de la solution à l'aide du langage Java	22
3.5.1 Algorithme de la boucle de simulation	22
3.5.2 Algorithme de calcul de l'état futur d'une cellule	23
3.5.3 Algorithme de calcul de la probabilité de propagation	24
Conclusion	26
Bibliographie	27
Annexes	A1

A Autre diagramme de classe proposé par l'équipe 1 A1

B Autre diagramme de classe proposé par l'équipe 2 A2

Table des figures

Figure 3.1 Voisinage de Von Neumann	9
Figure 3.2 Diagramme de Gantt du projet	13
Figure 3.3 Diagramme de Gantt du projet	13
Figure 3.4 Stratégie des branches et historique des commit	14
Figure 3.5 Diagramme de classes initial sélectionné par l'équipe	17
Figure 3.6 Version finale du diagramme de classes	18
Figure 3.7 Design Figma de l'interface graphique	20
Figure 3.8 Interface graphique réellement implémenté	21
Figure 3.9 Code de la méthode Grille.simulation()	23
Figure 3.10 Code de la méthode Arbre.calculeEtatFutur()	24
Figure 3.11 Code de la méthode Arbre.calculeP()	25
Figure A.1 Version finale du diagramme de classes	A1
Figure B.1 Autre diagramme de classe proposé par l'équipe 2	A2

Liste des tableaux

Tableau 3.1	Types de cellules	8
-------------	-------------------	-------	---

Introduction

L'augmentation régulière de la surface des forêts dévastées par les feux ces dernières années met en lumière l'urgence d'apporter des solutions pour mieux prévenir et gérer ces catastrophes naturelles. En France, l'année 2022 par exemple a été marquée par un record de surface végétative brûlée, avec plus de 62 000 hectares de forêt détruits lors de vagues de chaleur estivales. Face à l'urgence de ce phénomène qui ne cesse de s'aggraver chaque année, il est primordial de disposer d'outils permettant de mieux comprendre et maîtriser la propagation des feux de forêt.

Dans ce contexte, la simulation de la propagation des incendies de forêt peut offrir une solution prometteuse pour mieux comprendre et anticiper l'expansion de ces feux. Ce projet a pour objectif de proposer une interface graphique en langage Java pour simuler la propagation d'un feu de forêt en fonction des règles définies dans un automate cellulaire. L'interface permettra à l'utilisateur de configurer la parcelle de forêt et les positions initiales des arbres en feu, puis de visualiser l'évolution de l'incendie selon les règles de transition de l'automate cellulaire.

Ce rapport détaillera l'ensemble des étapes nécessaires à la réalisation de cette simulation, en passant en revue les concepts théoriques des automates cellulaires, le voisinage de Von Neumann, la modélisation mathématique et informatique du problème, ainsi que la gestion de projet et la conception de l'interface graphique. Nous explorerons également l'impact de différents facteurs sur la propagation du feu, tels que la direction du vent, l'humidité du sol, le climat et le type de végétation. Enfin, nous discuterons des différentes améliorations possibles de notre simulation, en explorant d'autres facteurs qui influencent le feu de forêt et en proposant des pistes pour intégrer des technologies plus avancées, telles que l'intelligence artificielle.

1 Revue de la littérature

La revue de la littérature effectuée dans le cadre de ce rapport a permis de recenser plusieurs études portant sur la modélisation de la propagation des incendies de forêt à l'aide des automates cellulaires. Parmi ces études, nous avons identifié neuf articles scientifiques qui ont retenu notre attention. Ces études présentent différentes approches et méthodologies pour la modélisation de la propagation des incendies de forêt, en utilisant les automates cellulaires comme outil de simulation. Les articles identifiés incluent : Hernandez Encinas et al. (2006) [1], Karafyllidis et Thanailakis (1996) [2], Alexandridis et al. (2008) [3], Berjak et Hearne (2001) [4], Drossel et Schwabl (1992) [5], Ghisu et al. (2015) [6], Mutthulakshmia et al. (2020) [7], Freire et DaCamara (2019) [8] et Hernandez Encinas et al. (2006) [9].

L'étude de Hernandez Encinas et al. (2006) [1] présente un modèle de propagation des incendies forestiers utilisant des automates cellulaires hexagonaux. Les auteurs ont intégré des paramètres tels que la densité des arbres, la vitesse et la direction du vent, ainsi que la présence de routes et de cours d'eau pour simuler la propagation des incendies. Les résultats montrent une corrélation significative entre les conditions environnementales et la vitesse de propagation de l'incendie.

L'étude de Karafyllidis et Thanailakis (1996) [2] a proposé un modèle de prédiction de la propagation des incendies forestiers en utilisant des automates cellulaires à deux dimensions. Le modèle a été développé en tenant compte des caractéristiques de la végétation, des conditions météorologiques et topographiques, ainsi que des comportements humains. Les résultats ont montré une corrélation significative entre la vitesse de propagation des incendies et la densité de la végétation. Cette étude a utilisé un voisinage de Moore pour la propagation de l'incendie.

L'étude de Alexandridis et al. (2008) [3] a présenté un modèle de prédiction de la propagation des incendies forestiers à l'aide d'automates cellulaires à deux dimensions. Le modèle a été développé pour simuler l'incendie qui a balayé l'île de Spetses en 1990. Les auteurs ont inclus des paramètres tels que la densité de la végétation, la direction et la vitesse du vent, ainsi que l'influence des facteurs topographiques. Les résultats ont montré une corrélation significative entre la vitesse de propagation de l'incendie et la densité de la végétation ainsi que la direction et la vitesse du vent.

L'étude de Berjak et Hearne (2001) [4] a présenté un modèle d'automate cellulaire amélioré pour simuler les incendies dans un système de savane spatialement hétérogène. Les auteurs ont pris en compte les effets des paramètres tels que la densité de la végétation, la direction et la vitesse du vent, la densité du combustible et la topographie. Les résultats ont montré une corrélation significative entre la vitesse de propagation de l'incendie et la densité de la végétation ainsi que les conditions météorologiques. Cette étude a utilisé un voisinage de von Neumann pour la propagation de l'incendie.

L'article de Drossel et Schwabl (1992) [5] introduit un modèle de propagation d'in-

cendie basé sur l'idée de la critique auto-organisée (self-organized criticality). Ce modèle suppose que les conditions de départ, comme la densité des arbres ou la présence d'herbes sèches, sont distribuées de manière aléatoire, ce qui conduit à une propagation d'incendie auto-organisée, c'est-à-dire que la propagation suit une loi de puissance plutôt que d'être régulière. Les auteurs ont également examiné la distribution spatiale de la taille des incendies, qui suit également une loi de puissance. Bien que ce modèle soit assez différent de celui que nous avons utilisé dans notre étude, il met en évidence l'importance des facteurs aléatoires et de la complexité des interactions dans la propagation des incendies.

Ghisu et al. (2015) [6] ont proposé quant à eux un algorithme optimisé pour la simulation de la propagation des incendies de forêt en utilisant un automate cellulaire testé sur un ensemble de données expérimentales recueillies sur le terrain. Les résultats ont montré que la simulation basée sur l'algorithme proposé est capable de reproduire la propagation réelle du feu avec une bonne précision. Les auteurs ont également comparé leur méthode avec d'autres approches d'automates cellulaires existantes et ont montré que leur algorithme est plus efficace et plus précis. Ce travail met en avant l'importance de l'optimisation des algorithmes pour la simulation de la propagation des incendies de forêt, ce qui peut être utile pour la gestion et la prévention des incendies de forêt. De plus, la méthodologie utilisée dans cet article peut être applicable à d'autres domaines qui utilisent des automates cellulaires pour simuler des phénomènes naturels.

L'étude de Mutthulakshmia et al. (2020) [7] propose une simulation de la propagation des feux de forêt à l'aide d'automates cellulaires basé sur des règles qui décrivent l'ignition, la propagation et l'extinction des feux. Ils ont également incorporé des règles pour la lutte contre les incendies, telles que l'utilisation de coupe-feu, d'eau et de produits chimiques. La simulation a été réalisée sur une zone de 1000x1000 cellules et les résultats ont été validés à l'aide de données satellitaires réelles. Les résultats ont montré que le modèle proposé est capable de reproduire avec précision les schémas de propagation du feu observés dans la réalité. Cette étude est intéressante car elle montre l'importance de prendre en compte des facteurs externes dans la modélisation des feux de forêt, tout en mettant en avant les avantages des automates cellulaires pour ce type de simulation.

Enfin, Les études menées respectivement par Freire et DaCamara (2019) [8], et par Hernandez Encinas et al. (2006) [9] utilisent elles aussi les automates cellulaires pour simuler la propagation des incendies de forêt. La première utilise une méthode de modélisation basée sur la propagation de la chaleur et sur la détermination des zones critiques. Les auteurs ont testé leur modèle en le comparant à des données réelles de feux de forêt et ont constaté que le modèle avait une précision satisfaisante. La deuxième étude quant à elle, utilise une approche similaire à celle de Hernandez Encinas et al. (2006) [1] en utilisant un voisinage de Von Neumann pour déterminer la probabilité de propagation de l'incendie de forêt. Les auteurs ont également proposé un algorithme optimisé pour améliorer la précision de la simulation.

Ces études ont toutes pour point commun l'utilisation des automates cellulaires pour modéliser la propagation des incendies de forêt. Elles diffèrent toutefois dans les

choix qu'elles font quant aux règles de propagation du feu. Certaines d'entre elles, telles que l'étude de Drossel et Schwabl (1992) [5], utilisent des règles relativement simples pour modéliser la propagation du feu, tandis que d'autres, comme l'étude de Ghisu et al. (2015) [6], ont recours à des règles plus complexes pour prendre en compte différents facteurs environnementaux.

Notre choix de nous concentrer sur le voisinage de Von Neumann et la prise en compte de facteurs externes dans le calcul de la probabilité de propagation est directement lié aux résultats et approches présentés dans ces études. En effet, plusieurs de ces études ont utilisé des approches similaires pour modéliser la propagation des incendies de forêt, en mettant l'accent sur l'influence des facteurs environnementaux et de la topographie dans la propagation du feu. Nous avons donc cherché à nous appuyer sur ces résultats pour réaliser une implémentation graphique d'un modèle simplifié de propagation des incendies de forêt à l'aide des automates cellulaires.

2 Quelques concepts théoriques

Dans cette partie, nous aborderons quelques concepts théoriques nécessaires à la compréhension de notre cas d'étude de simulation graphique d'un feu de forêt avec le langage Java. Nous commencerons par explorer le concept d'automate cellulaire, qui est une méthode de modélisation mathématique utilisée pour simuler des systèmes complexes tels que les feux de forêt. Ensuite, nous parlerons du diagramme de *Gantt*, un outil de gestion de projet qui nous permettra de planifier les différentes tâches à réaliser pour mener à bien notre projet. Nous aborderons également le langage de modélisation unifiée (UML) qui sera utilisé pour établir une modélisation informatique de notre projet. Nous verrons également l'importance de *Git*, un système de contrôle de version qui facilite le travail collaboratif sur un projet. Enfin, nous étudierons le langage de programmation *Java*, qui sera utilisé pour mettre en œuvre la simulation graphique du feu de forêt.

2.1 Automate cellulaire

Les automates cellulaires sont des modèles mathématiques permettant de simuler des phénomènes qui se déroulent dans des espaces discrets.[10] Ils sont largement utilisés dans la modélisation de systèmes complexes tels que la météorologie, l'écologie, la biologie et la physique. Dans notre cas d'étude, nous utiliserons les automates cellulaires pour simuler l'évolution d'un feu de forêt en fonction de différents paramètres tels que le vent, l'humidité, le type de végétation et le climat.

Le voisinage de *Von Neumann* est l'une des méthodes les plus courantes pour la modélisation de l'environnement à travers les automates cellulaires.[11] Cette méthode suppose que chaque cellule est en contact avec ses quatre voisins immédiats : en haut, en bas, à gauche et à droite. La règle de transition est alors appliquée à chaque cellule en fonction de l'état de ses voisins. Pour notre simulation de feu de forêt, le voisinage de *Von Neumann* nous permettra de prendre en compte l'influence des cellules environnantes sur le comportement du feu.

L'utilisation des automates cellulaires et du voisinage de *Von Neumann* nous permettra de simuler de manière réaliste l'évolution d'un feu de forêt et d'analyser l'impact de différents facteurs tels que la direction du vent, l'humidité du sol, le type de végétation et le climat sur la propagation du feu.

2.2 Diagramme de *Gantt*

Le diagramme de *Gantt* est un outil de gestion de projet utilisé pour visualiser les différentes tâches à effectuer dans le cadre d'un projet et leur durée respective. Il permet de planifier les différentes étapes du projet et de suivre l'avancement des tâches. Le diagramme de *Gantt* est un outil très utilisé dans le monde de l'entreprise et du management, mais il peut également être utile dans des projets personnels ou académiques [12].

Le diagramme de *Gantt* a été inventé par Henry *Gantt* au début du 20ème siècle et a été initialement utilisé pour planifier les activités de production dans les usines. Depuis

lors, il a été adapté pour être utilisé dans une grande variété de projets et d'industries. Le diagramme de *Gantt* est un outil simple mais efficace pour planifier les tâches et gérer les délais [13].

Dans le cadre de ce projet, nous avons utilisé le diagramme de *Gantt* pour planifier les différentes étapes de notre simulation de feu de forêt. Nous avons commencé par identifier les tâches à effectuer, telles que la modélisation mathématique du problème, la création de l'interface utilisateur, la mise en place de l'algorithme de simulation, etc. Ensuite, nous avons estimé la durée de chaque tâche et nous avons organisé ces tâches dans une chronologie cohérente. L'utilisation du diagramme de *Gantt* nous a permis de suivre l'avancement du projet et de nous assurer que nous étions en mesure de respecter les délais impartis. Nous avons également été en mesure de détecter des retards potentiels dans le projet et de prendre des mesures pour les éviter.

Dans l'ensemble, le diagramme de *Gantt* a été un outil très utile pour la gestion de notre projet de simulation de feu de forêt. Nous recommandons son utilisation pour toute personne impliquée dans la gestion de projets, qu'ils soient personnels ou professionnels.

2.3 UML

UML (Unified Modeling Language) est un langage de modélisation visuelle utilisé pour spécifier, visualiser, concevoir et documenter les artefacts d'un système logiciel.[14] UML est largement utilisé dans l'industrie du développement de logiciels pour représenter graphiquement les modèles conceptuels de systèmes logiciels. Les diagrammes UML représentent les différents aspects d'un système, tels que les exigences fonctionnelles, l'architecture logicielle, les interactions entre les composants, les processus métier, etc.

L'UML est donc un langage de modélisation très utilisé dans le domaine du développement de logiciels. Il est possible de créer plusieurs types de diagrammes UML, comme le diagramme de classes, le diagramme de cas d'utilisation, le diagramme de séquence, le diagramme de collaboration, etc. Cependant, pour notre simulation, nous nous sommes concentrés sur le diagramme de classes, qui permet de représenter les classes et leurs relations dans un système.

Le diagramme de classes est donc un outil essentiel pour décrire la structure d'un système orienté objet. Il permet de représenter les classes et leurs relations sous forme de boîtes reliées entre elles par des flèches, représentant les relations entre les classes. Les classes sont représentées par des rectangles, contenant le nom de la classe, les attributs et les méthodes de la classe. Les relations entre les classes sont représentées par des flèches, qui peuvent être unidirectionnelles ou bidirectionnelles. Les flèches unidirectionnelles indiquent que la relation ne va que dans un sens, tandis que les flèches bidirectionnelles indiquent que la relation va dans les deux sens.

2.4 Git

Git est un système de contrôle de version très populaire pour le développement de logiciels. Il permet à plusieurs développeurs de travailler ensemble sur un même projet, tout en conservant un historique de toutes les modifications apportées au code [15]. Git utilise une approche décentralisée, dans laquelle chaque développeur possède une copie locale du code source. Les modifications apportées par chaque développeur sont enregistrées localement, puis peuvent être partagées avec les autres développeurs via un référentiel Git centralisé. Git permet également de fusionner facilement les modifications apportées par différents développeurs, ce qui facilite la collaboration.

Pour notre projet de simulation de feu de forêt, nous avons utilisé Git pour gérer le code source et faciliter la collaboration entre les membres de l'équipe de développement. Nous avons créé un référentiel Git centralisé, sur lequel chaque membre de l'équipe a travaillé localement en utilisant une branche dédiée. Nous avons utilisé les commandes Git de base pour gérer les modifications apportées au code source, notamment pour ajouter, modifier, supprimer et fusionner les fichiers.

En utilisant Git, nous avons pu conserver un historique de toutes les modifications apportées au code source de notre projet de simulation de feu de forêt. Cela nous a permis de revenir à des versions précédentes du code en cas de problèmes, ainsi que de travailler simultanément sur différentes fonctionnalités sans risquer de perdre des modifications importantes.

2.5 Java

Java est un langage de programmation orienté objet largement utilisé dans le développement d'applications pour les ordinateurs, les téléphones mobiles et autres appareils électroniques. Il a été créé en 1995 par James Gosling chez Sun Microsystems, qui a depuis été acquis par Oracle. Java est devenu populaire en raison de sa portabilité, c'est-à-dire sa capacité à fonctionner sur plusieurs plates-formes, comme Windows, Mac OS et Linux. Cela est possible grâce à l'utilisation de la machine virtuelle Java (JVM), qui permet d'exécuter le code Java sur différents systèmes d'exploitation.

Java est souvent utilisé pour développer des applications de bureau, des applications Web et des jeux vidéo. Il dispose également d'une grande communauté de développeurs et de nombreuses bibliothèques open source, telles que Spring, Hibernate et Apache Struts, qui facilitent et accélèrent le développement d'applications.

En ce qui concerne le développement de la simulation de feu de forêt, Java est un choix judicieux car il permet de créer facilement une interface graphique pour afficher la simulation en temps réel. De plus, le langage est suffisamment puissant pour permettre une manipulation facile des automates cellulaires et des calculs mathématiques nécessaires à la simulation.

3 Cas d'étude : Simulation graphique d'un feu de forêt avec le langage Java

Dans cette partie, nous allons étudier un cas pratique de simulation graphique d'un feu de forêt en utilisant le langage de programmation Java. La simulation est basée sur des automates cellulaires et prend en compte plusieurs paramètres tels que le voisinage de *Von Neumann*, la direction du vent, l'humidité du sol, le type de végétation, et les conditions climatiques. Nous allons explorer la modélisation mathématique du problème, la gestion de projet, la modélisation informatique, la conception de l'interface utilisateur et la mise en œuvre de la solution. Cette étude permettra de mieux comprendre l'application des automates cellulaires dans la simulation de phénomènes naturels et de développer des compétences en programmation Java.

3.1 Modélisation mathématique du problème

La modélisation mathématique du problème de propagation du feu dans une forêt est une étape importante pour comprendre les mécanismes sous-jacents et pour prédire l'évolution de l'incendie. Cette partie vise à décrire la modélisation mathématique que nous avons utilisée pour simuler la propagation du feu dans notre système. Nous avons utilisé une approche basée sur une grille de cellules où chaque cellule peut avoir trois états différents : *sol nu*, *arbre vivant* et *arbre en feu*. Nous avons également défini des règles de propagation qui prennent en compte différents facteurs tels que le *voisinage*, la *direction du vent*, la *saison*, l'*humidité du sol* et le *type de végétation*. Nous avons implémenté ces règles dans un modèle mathématique qui nous permet de simuler la propagation du feu dans la forêt et d'étudier les effets de chaque facteur sur la propagation de l'incendie.

La modélisation de la forêt a été effectuée à l'aide d'une grille de taille $n \times n$ contenant n^2 cellules. Chaque cellule peut avoir quatre états possibles tels que décrit dans le tableau 3.1. Des règles de propagation ont été établies afin de déterminer l'état futur de chaque cellule en fonction de son état actuel et de l'état de ses voisins. Si la cellule est un arbre en feu, son état futur est "arbre en cendre". Si elle est un arbre vivant et qu'aucun de ses voisins n'est en feu, alors son état futur est le même que son état actuel. Si elle est un arbre vivant et au moins un de ses voisins est en feu, alors sa probabilité de prendre feu est calculée. La probabilité initiale est fixée à 0.6, ce qui correspond à une probabilité raisonnable pour un incendie de forêt.

Type de cellule	Couleur de la cellule	Code
Sol nu	Ivoire	0
Arbre vivant	Vert	1
Arbre en feu	Rouge	2
Arbre en cendre	Gris	3

Table 3.1 – Types de cellules

Cependant, cette probabilité fixe peut ne pas être réaliste dans toutes les situations. C'est pourquoi des facteurs externes ont été intégrés dans le modèle afin d'améliorer sa précision. Ces facteurs incluent le voisinage, la direction du vent, la saison, l'humidité du sol et le type de végétation.

3.1.1 Impact du voisinage

Dans notre modèle de simulation de la propagation d'un incendie forestier, le voisinage d'une cellule est un élément important qui peut avoir un impact significatif sur l'état de la cellule elle-même. Le voisinage de *Von Neumann* a été choisi pour notre simulation, car il permet de prendre en compte les cellules voisines situées dans les quatre directions cardinales (nord, sud, est, ouest). Le voisinage de *Von Neumann* est défini comme l'ensemble des cellules situées à une distance de *Manhattan* égale à 1 de la cellule en question. Autrement dit, si nous considérons une cellule donnée dans notre grille, son voisinage de *Von Neumann* est composé de ses quatre voisins directs situés au nord, au sud, à l'est et à l'ouest. La figure 3.1 présente de façon graphique ce voisinage.

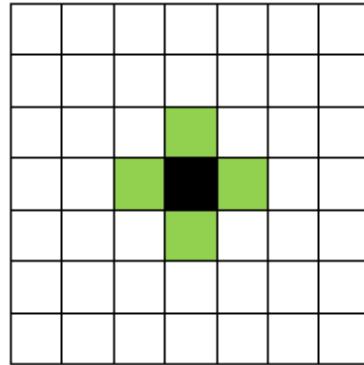


Figure 3.1 – Voisinage de Von Neumann

Lorsqu'une cellule vivante a au moins un voisin en feu, nous calculons la probabilité pour qu'elle aussi s'enflamme. Partant de la probabilité de base p initialement fixée à 0.6, la formule 1 est appliquée pour mettre à jour celle-ci de sorte qu'elle prenne en compte le nombre n de voisins en feu de la cellule.

$$p = p + (n \times 0.05) \quad (1)$$

Le voisinage de Von Neumann peut avoir un impact significatif sur la simulation de la propagation de l'incendie forestier, car si plusieurs cellules sont en feu dans le voisinage direct d'une cellule donnée, la probabilité pour qu'elle s'enflamme à son tour augmente considérablement. Cependant, si la cellule a des voisins qui ne sont pas en feu, cela peut diminuer la probabilité qu'elle aussi s'enflamme. En prenant en compte le voisinage de Von Neumann dans notre simulation, nous pouvons avoir une représentation plus réaliste de la propagation de l'incendie forestier. Cela nous permet également d'explorer les différents facteurs qui peuvent influencer cette propagation et d'ajuster notre modèle en conséquence.

3.1.2 Impact de la direction du vent

L'impact de la direction du vent est un facteur important qui peut influencer la propagation du feu dans la forêt. Dans notre modélisation, nous avons pris en compte la direction du vent pour déterminer la probabilité qu'une cellule vive prenne feu en fonction de ses voisins. La direction du vent peut être définie par rapport à quatre points cardinaux : le nord, le sud, l'est et l'ouest. Si le vent souffle dans la direction opposée à celle d'une cellule en feu, alors la probabilité pour que cette cellule en vie prenne feu diminue. En effet, le vent peut empêcher la propagation du feu en poussant la fumée et les flammes dans la direction opposée. Cependant, si le vent souffle dans la même direction que celle d'une cellule en feu, alors la probabilité pour que cette cellule en vie prenne feu augmente. Le vent peut en effet alimenter le feu en oxygène, augmentant ainsi la probabilité que le feu se propage.

Dans notre modélisation, nous avons déterminé l'impact de la direction du vent en utilisant une valeur binaire b qui indique si le vent pousse le feu vers la cellule en question. Si $b = 1$, cela signifie que le vent pousse le feu dans la direction de la cellule en vie, et donc la probabilité que cette cellule prenne feu est augmentée de 0.5. En revanche, si $b = 0$, cela signifie que le vent souffle dans une direction opposée à celle de la cellule en vie, et donc la probabilité que cette cellule prenne feu est diminuée de 0.1. La formule 2 résume l'impact que la direction du vent a sur la probabilité de propagation du feu.

$$p = \begin{cases} p + 0.5 & \text{si } b = 1 \\ p - 0.1 & \text{si } b = 0 \end{cases} \quad (2)$$

Notons que l'utilisateur peut également décider d'ignorer l'impact du vent en choisissant *INDIFFERENT* comme direction du vent. Dans ce cas, la probabilité pour qu'une cellule en vie prenne feu sera déterminée uniquement en fonction de l'état de ses voisins, sans tenir compte de la direction du vent.

En conclusion, l'impact de la direction du vent est un facteur important à prendre en compte pour modéliser la propagation du feu dans la forêt. Dans notre modélisation, nous avons intégré cet impact en augmentant ou diminuant la probabilité qu'une cellule en vie prenne feu en fonction de la direction du vent et de la position des cellules en feu. Cela permet de créer une simulation plus réaliste.

3.1.3 Impact du climat

Les saisons ont un impact important sur la propagation des feux de forêt. En effet, les variations de température et d'humidité peuvent changer considérablement la probabilité de propagation d'un feu dans une forêt. Dans notre modèle, nous avons donc décidé d'intégrer cet impact en modifiant la probabilité initiale pour chaque cellule. Tout d'abord, en hiver, les températures sont basses et l'humidité est souvent plus élevée, ce qui réduit la probabilité qu'un feu se propage. Ainsi, nous avons choisi de diminuer la probabilité initiale de prendre feu de 10% pour les arbres pendant cette saison (voir formule 3).

$$p = p - 0.1 \quad (3)$$

En été, les températures sont élevées et l'humidité est généralement plus faible, ce qui favorise la propagation des feux. Nous avons donc choisi d'augmenter la probabilité initiale de prendre feu de 20% pour les arbres pendant cette saison (voir formule 3).

$$p = p + 0.2 \quad (4)$$

Pour le printemps et l'automne, nous avons considéré que les conditions étaient relativement neutres et n'avaient donc pas d'impact significatif sur la probabilité de propagation d'un feu. Nous avons donc gardé la probabilité initiale de prendre feu inchangée pour ces deux saisons.

En intégrant l'impact des saisons sur la probabilité initiale de prendre feu, nous avons pu rendre notre modèle plus réaliste et plus adapté à la réalité de la propagation des feux de forêt dans différentes conditions météorologiques. Cela permet également d'explorer différents scénarios pour les feux de forêt dans différentes saisons, ce qui peut être utile pour la planification des mesures de prévention et de lutte contre les feux de forêt.

3.1.4 Impact de l'humidité du sol

L'humidité du sol joue également un rôle important dans la propagation des incendies de forêt. En effet, un sol humide peut aider à ralentir ou même à éteindre les flammes. Cela est dû au fait que l'eau est un excellent conducteur de la chaleur et qu'elle peut absorber une grande quantité de chaleur avant de s'évaporer. Par conséquent, un sol humide peut aider à réduire la température autour d'un arbre en feu et ainsi ralentir la propagation du feu.

Dans notre modèle, nous avons intégré cet effet en réduisant la probabilité qu'un arbre prenne feu lorsque le sol est humide. Plus précisément, lorsque la cellule est humide ou encore lorsque le sol sur lequel est placé l'arbre est humide, la probabilité initiale de prendre feu est réduite de 0,1. Cela signifie que même si un arbre a des voisins en feu et que les autres facteurs négatifs ont été pris en compte, la probabilité qu'il prenne feu sera réduite de 0,1 si le sol est humide. Cet effet peut avoir un impact significatif sur la propagation des incendies de forêt. Par exemple, si une zone a connu de fortes pluies et que le sol est humide, cela peut aider à ralentir la propagation des flammes en réduisant la probabilité qu'un arbre prenne feu. En revanche, si une zone est en proie à une sécheresse prolongée et que le sol est sec, cela peut aggraver la propagation des incendies en augmentant la probabilité qu'un arbre prenne feu.

En conclusion, l'humidité du sol est un facteur important à prendre en compte dans la modélisation des incendies de forêt, car elle peut avoir un impact significatif sur la propagation des flammes. En intégrant cet effet dans notre modèle, nous avons pu mieux simuler la propagation des incendies en fonction des conditions météorologiques et environnementales locales.

3.1.5 Type de végétation

Le type de végétation est un autre facteur important qui peut affecter la propagation du feu dans une forêt. Certaines espèces d'arbres sont plus résistantes au feu que d'autres. Par exemple, les conifères, comme les pins et les sapins, ont tendance à brûler plus facilement que les feuillus comme les chênes et les érables. Les arbres qui ont une écorce épaisse et résistante sont également moins vulnérables aux flammes que ceux dont l'écorce est fine. Dans notre modèle, nous avons introduit une variable booléenne *arbre peu inflammable* pour représenter la résistance au feu des arbres. Si cette variable est vraie, alors la probabilité pour qu'un arbre prenne feu est réduite. Nous avons fixé une valeur de -0,2 à cette probabilité, ce qui signifie qu'un arbre peu inflammable a 20% de chances en moins de prendre feu qu'un arbre ordinaire.

L'impact du type de végétation sur la propagation du feu peut varier considérablement selon l'environnement et les conditions météorologiques. Les arbres qui poussent dans des zones humides ou qui ont accès à des sources d'eau souterraines sont souvent plus résistants aux incendies que ceux qui poussent dans des régions arides ou soumises à une sécheresse prolongée. Dans notre modèle, nous avons également tenu compte de l'humidité du sol pour refléter cet impact potentiel.

En somme, le choix du type de végétation peut jouer un rôle crucial dans la propagation du feu dans une forêt. En modifiant la résistance au feu des arbres, nous pouvons simuler différents types de forêts et étudier leur comportement face aux incendies. Cela permettra de mieux comprendre comment la composition de la forêt affecte la probabilité de propagation du feu et aidera à élaborer des stratégies de prévention et de lutte contre les incendies.

3.2 Gestion de projet

La gestion de projet est un aspect essentiel de tout travail collaboratif. Dans notre projet de modélisation d'incendies de forêt, nous avons utilisé plusieurs outils de gestion de projet pour nous aider à rester organisés, à suivre les progrès et à gérer les tâches de manière efficace. Les quatre technologies que nous avons utilisées sont le diagramme de *Gantt*, *Trello*, *Git* et la méthodologie *agile*.

3.2.1 Diagramme de Gantt

Le diagramme de Gantt est un outil de gestion de projet utilisé pour planifier et suivre les tâches à accomplir dans un projet. Il permet de visualiser les tâches, leur durée et leur dépendance les unes par rapport aux autres. Cet outil a été très utile pour notre projet car il nous a permis de suivre le progrès et de voir les tâches restantes à accomplir. Nous avons pu identifier les tâches qui prenaient plus de temps que prévu et ajuster notre planification en conséquence. Le diagramme de Gantt a également aidé à définir les rôles et les responsabilités de chaque membre de l'équipe. La figure 3.2 présente une capture ponctuelle du diagramme de Gantt que nous avons utilisé.

Simulation_feu_de_foret

17 mars 2023

Diagramme de Gantt

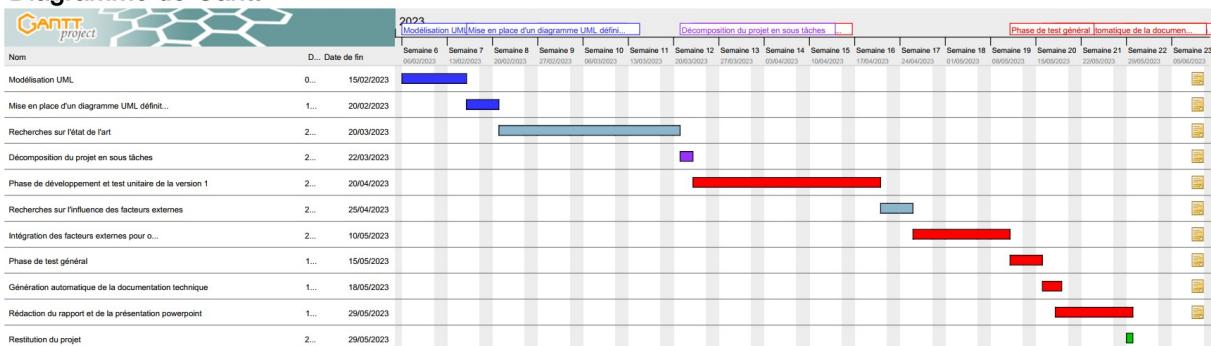


Figure 3.2 – Diagramme de Gantt du projet

3.2.2 Trello

Trello est un outil de gestion de projet basé sur des tableaux kanban. Il permet de créer des listes de tâches, des cartes pour chaque tâche, et des étiquettes pour classer les tâches. Nous avons utilisé Trello pour suivre les tâches quotidiennes, affecter des membres de l'équipe à des tâches spécifiques, et pour garder une trace des progrès de chaque tâche. Cela nous a permis de nous concentrer sur les tâches les plus importantes et de les terminer rapidement. Nous avons également pu ajouter des commentaires et des notes pour que chaque membre de l'équipe reste informé de l'avancement du projet. La figure 3.3 présente une capture ponctuelle de notre tableau Trello présentant les différentes tâches ainsi que leurs statuts.

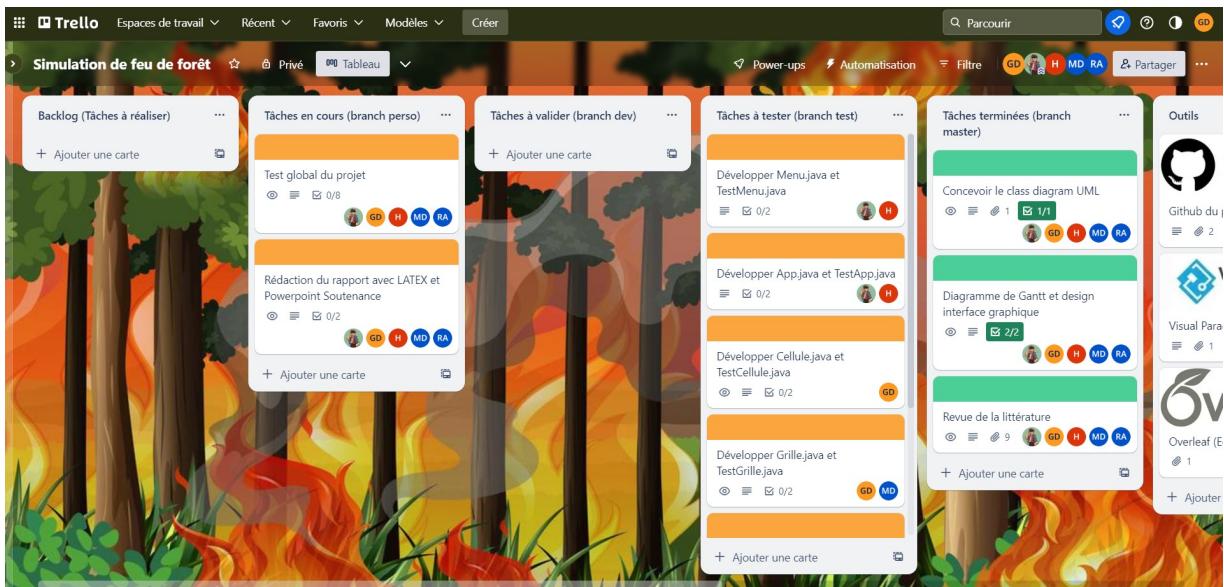


Figure 3.3 – Diagramme de Gantt du projet

3.2.3 Git

Git est un outil de gestion de version pour le code source. Il permet de suivre les modifications du code, de restaurer les versions précédentes, et de collaborer sur le code avec plusieurs membres de l'équipe. Nous avons utilisé Git pour notre code de modélisation d'incendies de forêt [16], ce qui a facilité la collaboration entre les membres de l'équipe. Nous avons pu travailler sur différentes parties du code en même temps sans craindre de perdre des modifications importantes. Nous avons également utilisé les branches de Git pour développer de nouvelles fonctionnalités sans interrompre le développement actuel. Plus concrètement, les branches suivantes ont été créées :

1. Une branche *master* pour la version finale du code
2. Une branche *test* pour les fonctionnalités prêtes à être validées par les membres de l'équipe
3. Une branche *dev* pour les fonctionnalités consolidées mais qui doivent être intégrées au projet dans son ensemble
4. Une branche pour chaque membre d'équipe pour le développement proprement dit des tâches.

La figure 3.4 présente une représentation graphique de la stratégie de branches que nous avons adopté ainsi qu'un aperçu ponctuel de l'historique des commit.

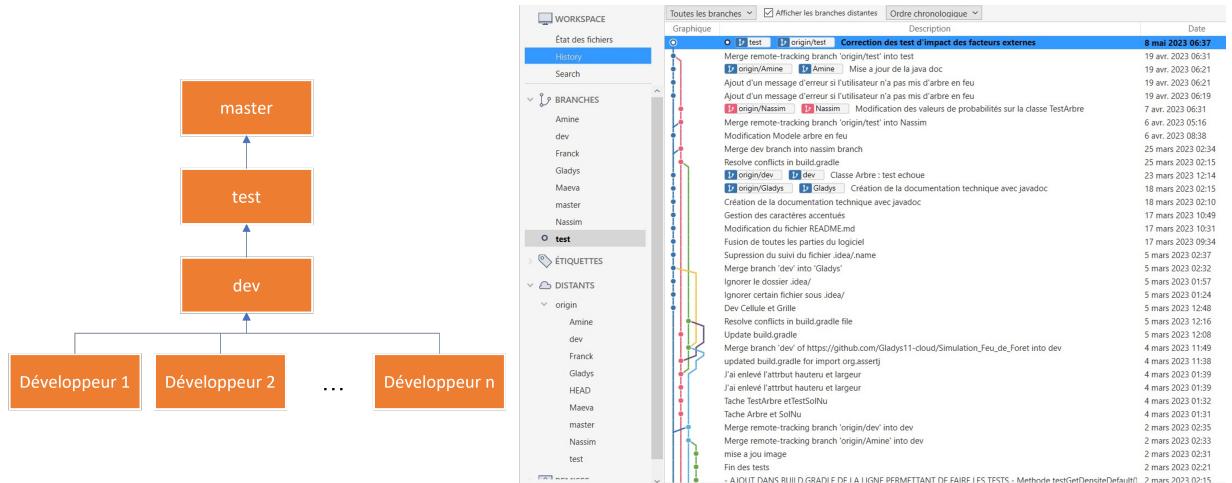


Figure 3.4 – Stratégie des branches et historique des commit

3.2.4 Méthodologie Agile

Nous avons opté pour la méthodologie agile dans la gestion de notre projet de modélisation d'incendie de forêt. Cette méthode de gestion de projet est basée sur des itérations courtes appelées sprints, qui durent généralement entre une et quatre semaines. Chaque sprint a un objectif spécifique et des tâches sont assignées aux membres de l'équipe pour atteindre cet objectif. Cette méthode a plusieurs avantages par rapport aux méthodes de gestion de projet traditionnelles.

Tout d'abord, l'approche agile permet une collaboration étroite entre les membres de l'équipe et les parties prenantes du projet. Les parties prenantes sont impliquées dès le début du projet et peuvent donner leur avis sur les priorités et les fonctionnalités à inclure dans chaque sprint. Cela signifie que les membres de l'équipe peuvent répondre rapidement aux changements de priorités ou aux commentaires des parties prenantes, ce qui garantit que le projet reste aligné sur les objectifs de l'entreprise. Ensuite, l'approche agile permet une meilleure visibilité sur la progression du projet. Les sprints sont divisés en tâches plus petites et plus gérables, ce qui signifie que l'équipe peut suivre de près l'avancement du projet et savoir exactement où en est chaque tâche. Cela permet à l'équipe de réagir rapidement en cas de retard ou de problème dans une tâche, ce qui minimise les risques pour le projet dans son ensemble. Enfin, l'approche agile encourage l'itération et l'amélioration continue. Après chaque sprint, l'équipe peut prendre en compte les commentaires des parties prenantes et les leçons apprises pendant le sprint pour améliorer le projet dans le prochain sprint. Cette approche itérative permet une adaptation continue du projet, ce qui garantit que le résultat final est le plus proche possible des besoins et des attentes des parties prenantes.

Pour faciliter la gestion de notre projet agile, nous avons utilisé plusieurs outils. Le diagramme de Gantt a été utilisé pour planifier les sprints et les tâches associées, en visualisant la durée et les dépendances entre les tâches. Trello a été utilisé pour suivre l'état de chaque tâche et de chaque sprint, permettant ainsi à l'équipe de savoir exactement où en est chaque tâche et de faciliter la collaboration entre les membres de l'équipe. Git a été utilisé pour gérer le code source de notre projet, ce qui a permis à l'équipe de travailler simultanément sur différentes parties du code et de suivre les modifications apportées au code.

En somme, la méthodologie agile s'est avérée très utile dans la gestion de notre projet de modélisation d'incendie de forêt. Elle nous a permis de travailler de manière collaborative, de suivre de près l'avancement du projet et de faire des améliorations continues pour répondre aux besoins qui ont évolués au fil du temps. Les outils que nous avons utilisés, comme le *diagramme de Gantt*, *Trello* et *Git*, ont permis à l'équipe de travailler de manière efficace et coordonnée. L'utilisation de ces outils a grandement amélioré notre efficacité en tant qu'équipe.

3.3 Modélisation informatique du problème

La modélisation informatique est une étape cruciale dans la conception d'un logiciel. Elle permet de représenter les différentes composantes du système ainsi que les interactions entre elles. Dans ce projet, nous avons utilisé le diagramme de classe de l'UML pour modéliser notre système de simulation de feu de forêt. Ce diagramme nous a permis de visualiser les différentes classes du système, les relations entre elles ainsi que les attributs et les méthodes de chaque classe. De plus, cette modélisation nous a permis de mieux comprendre le fonctionnement de notre système et ainsi de faciliter la phase d'implémentation.

3.3.1 Construction de la version initiale du diagramme de classes

La construction du diagramme de classe UML est une des stratégies les plus répandues de modélisation informatique d'un projet. Dans notre cas, la modélisation de la simulation d'incendie de forêt ne faisait pas exception. Afin de se partager les tâches et que chacun puisse visualiser la structure du projet, nous avons décidé de travailler individuellement sur des diagrammes de classes distincts. L'idée était de mettre en place chacun une représentation personnelle du projet tel que nous le visualisions. L'objectif était ensuite de récupérer les meilleures idées et de les intégrer pour réaliser le diagramme final.

Au cours de cette phase de modélisation, nous avons passé beaucoup de temps à discuter de la structure et de l'organisation du projet. Nous avons tous apporté des idées et des suggestions pour améliorer le modèle. Nous avons également cherché à intégrer toutes les fonctionnalités du projet, en nous appuyant sur les bonnes pratiques de la modélisation orientée objet. En annexe [lien vers annexe] à ce rapport, nous présentons quelques-uns des diagrammes de classes proposés par l'équipe.

Finalement, après des entretiens et en prenant en compte le retour de tous les membres d'équipe, nous avons sélectionné le diagramme de classe UML qui était le plus complet et intuitif. Cette version initiale nous a permis d'avoir une représentation claire et précise des différentes classes et de leurs interactions. Elle a également servi de base pour l'implémentation de notre programme de simulation d'incendie de forêt. La figure 3.5 présente le diagramme de classes initial sélectionné par l'équipe.

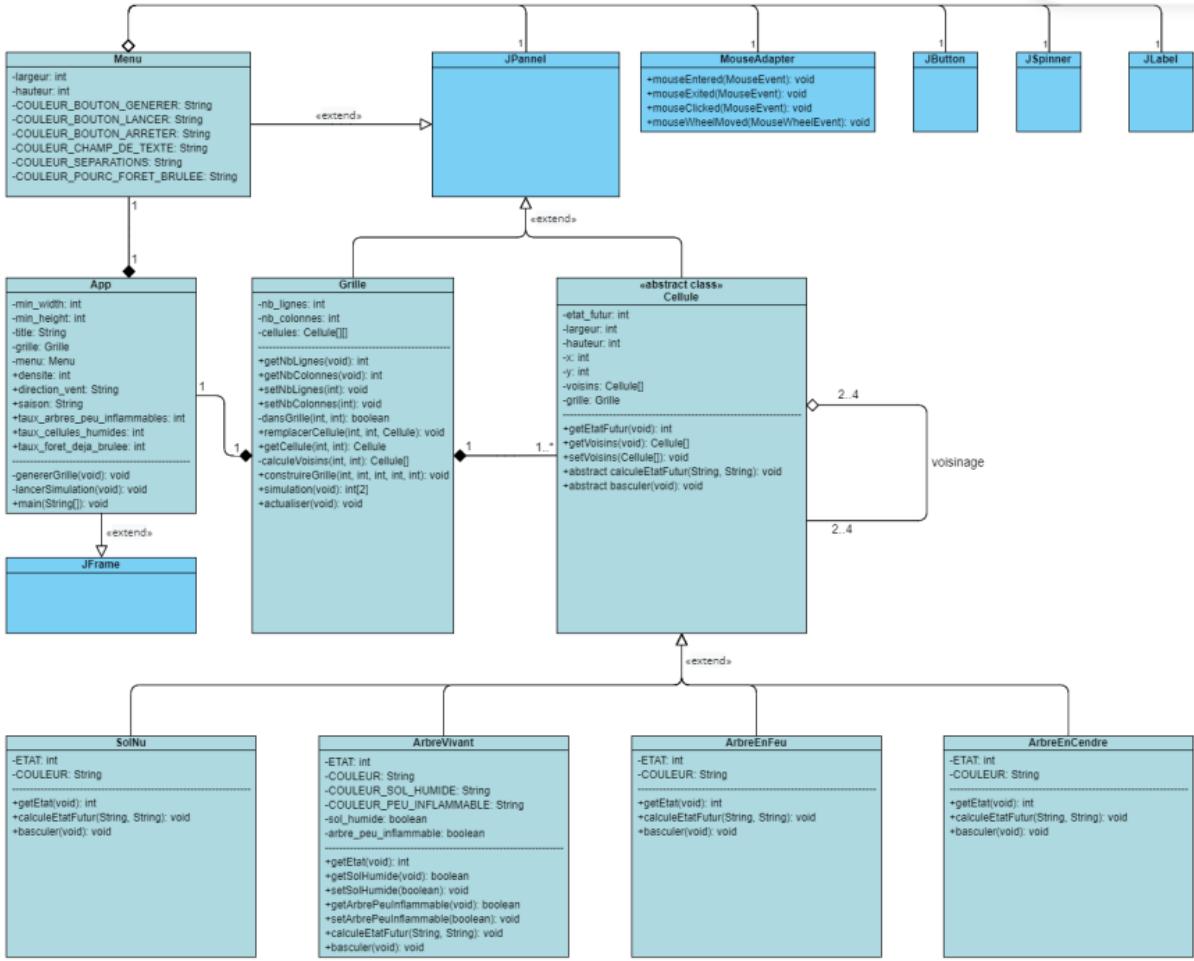


Figure 3.5 – Diagramme de classes initial sélectionné par l'équipe

3.3.2 Amélioration du diagramme

L'amélioration du diagramme de classes initial a été une étape importante pour s'assurer de la qualité de notre projet. En effet, le premier diagramme était déjà une bonne base de départ, mais nous avons réalisé qu'il était possible de le simplifier encore plus. Nous avons donc examiné les différentes classes et les relations entre elles, afin de voir où nous pourrions faire des améliorations. L'un des points clés de cette amélioration était de s'assurer que notre implémentation respecte les principes SOLID, qui sont des règles de base en programmation pour assurer une bonne qualité de code. Nous avons donc pris en compte ces principes lors de la révision du diagramme de classes.

Un des choix que nous avons fait pour simplifier le diagramme a été de regrouper les différentes classes d'arbres en une seule classe Arbre, dont l'état est déterminé par un attribut. Cette modification a permis de réduire le nombre de classes nécessaires dans le diagramme, ce qui a rendu le tout plus facile à comprendre et à implémenter. Après discussion et prise en compte des retours de chacun, ce diagramme a été adopté comme version finale.

Cette amélioration du diagramme de classes nous a permis d'obtenir une structure plus simple et plus claire pour notre projet, tout en respectant les principes SOLID. Grâce à cela, nous avons abouti à un code plus facile à comprendre et à une implémentation plus efficace. La figure 3.6 présente la version finale du diagramme de classes qui a été adopté par l'équipe.

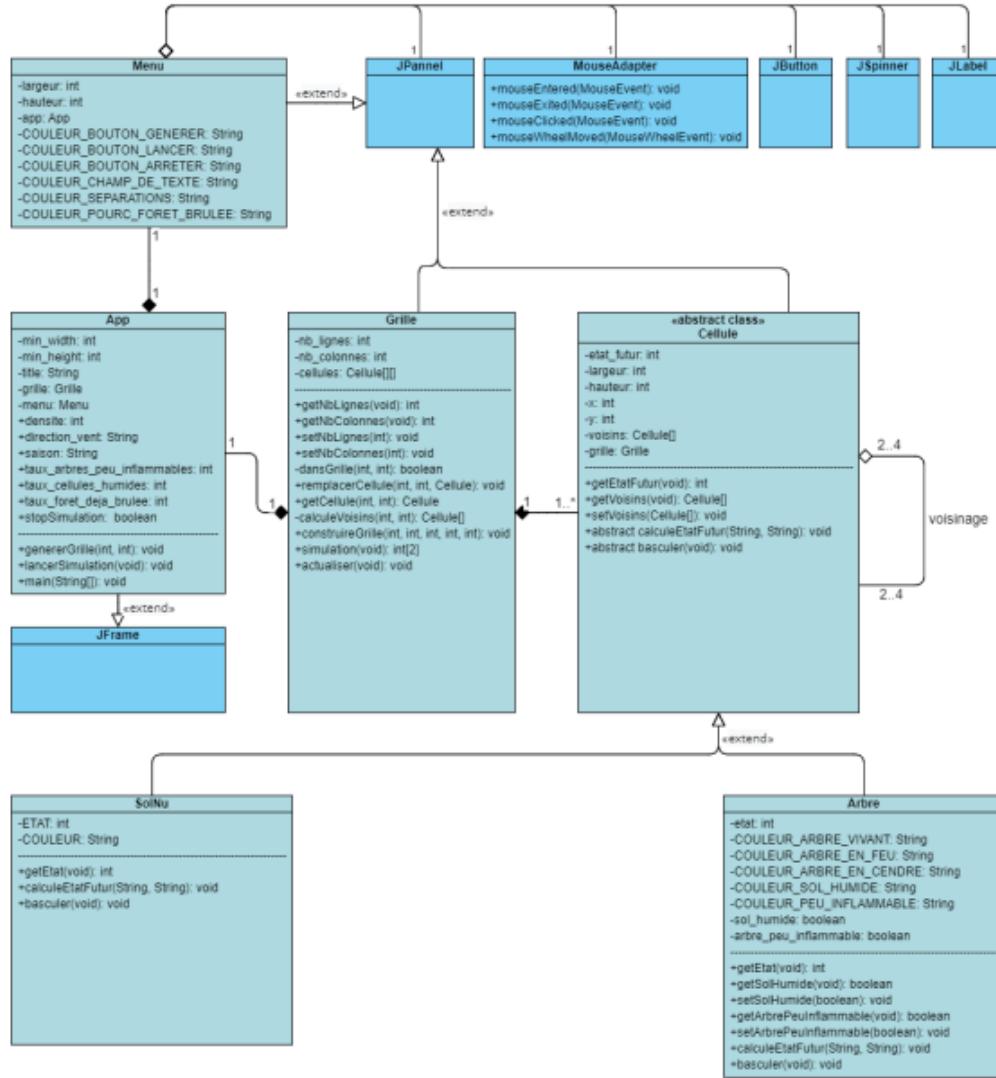


Figure 3.6 – Version finale du diagramme de classes

3.3.3 Description du diagramme de classes

Le diagramme de classes final comprend quatre classes principales et deux autres qui en spécialisent une. Ces classes sont les suivantes : **App**, **Menu**, **Grille**, **Cellule**, **SolNu** et **Arbre**. Nous allons expliquer le rôle de chacune de ces classes dans le modèle ainsi que les fonctions remplies par certaines de leurs méthodes.

1. Classe **App** : La classe **App** est la fenêtre principale du projet, point de départ de l'application. C'est une extension de la classe **JFrame** de java. Elle contient notamment la méthode **main** qui permet d'exécuter l'ensemble du programme. Le constructeur de la classe permet de créer la fenêtre de l'application et d'y ajouter la grille de simulation et le menu. Cette classe possède également plusieurs attributs tels que la grille, le menu, la densité des arbres, la direction du vent, la saison, le taux d'arbres peu inflammables, le taux de cellules humides, le taux de forêt déjà brûlée et un booléen qui permet de savoir si l'utilisateur a cliqué sur le bouton *Arrêter*. Elle a également des méthodes qui permettent de générer la grille et de lancer la simulation pour la mettre à jour de façon itérative jusqu'à ce qu'il n'y ait plus de changement dans la grille ou que l'utilisateur clique sur le bouton *Arrêter*.
2. Classe **Menu** : La classe **Menu** est une extension de la classe **JPanel** de java et contient quant à elle tous les composants graphiques tels que les boutons, les champs de texte et les lignes de séparation qui permettent de configurer la grille et de lancer la simulation. Toutes les interactions de l'utilisateur avec le programme sont prises en charge par cette classe. La classe **Menu** a des attributs tels que la largeur et la hauteur minimale du panneau, ainsi que des couleurs en hexadécimal pour les boutons et les champs de texte.
3. Classe **Grille** : La classe **Grille** est également une extension de **JPanel** et permet d'implémenter la grille sur laquelle se déroulera la simulation. Elle a des attributs tels que le nombre de lignes et de colonnes de la grille, ainsi qu'un tableau à deux dimensions contenant toutes les cellules de la grille. Elle a également des méthodes qui permettent de construire la grille et de mettre à jour la grille lors de la simulation. De par sa fonction, elle possède des méthodes utiles permettant d'accéder à une cellule spécifique de la grille, d'en déterminer les voisins et même de la modifier. Notons également que depuis l'interface, l'utilisateur pourra cliquer sur des cellules pour changer leur état et ainsi définir les arbres initialement en feu pour ensuite lancer une simulation.
4. Classe **Cellule** : La classe **Cellule** est une classe abstraite qui étend **JPanel** et qui représente les différentes composantes de la forêt, telles que le sol nu, l'arbre vivant, l'arbre en feu et l'arbre en cendres. Nous pouvons considérer chaque cellule de la grille comme représentant $1m^2$ de la forêt. La classe possède une méthode centrale du programme à savoir **abstract void calculeEtatFutur (String direction_vent, String saison)**. Cette méthode abstraite est définie par les deux classes concrètes **SolNu** et **Arbre** qui héritent de **Cellule**. C'est une méthode très importante car elle permet de calculer l'état futur d'une cellule en se basant sur plusieurs critères tels que présentés dans la section 3.1. Plus de détails sur cet algorithme seront donnés dans la section dédiée 3.5.2.
5. Classe **SolNu** : Cette classe hérite de la classe **Cellule** et permet de modéliser un sol nu, ce qui correspond à une parcelle de terrain sans arbre. Sa méthode principale est **abstract void calculeEtatFutur (String direction_vent, String**

saison) qu'elle hérite de `Cellule`. Cependant, la méthode retourne simplement l'état courant de la cellule car un sol nu ne change pas d'état après le passage du feu.

6. Classe `Arbre` : Cette classe hérite elle aussi de la classe `Cellule`. Elle permet de modéliser un arbre. Ses méthodes permettent de construire des arbres ayant des caractéristiques différentes afin de pouvoir observer le comportement du feu faces à divers types de forêts. C'est ainsi que l'utilisateur pourra depuis l'interface graphique décider de l'humidité du sol sur lequel repose les racines de l'arbres, et de l'inflammabilité de l'arbre. Tout comme `SolNu`, cette classe implémente la méthode `calculeEtatFutur()` dont le code est présenté dans la section 3.5.2.

3.4 Design de l'interface graphique

Le design de l'interface graphique est une étape phare du développement d'un logiciel car il permet de concevoir visuellement la façon dont les utilisateurs vont interagir avec l'application. Cela permet également de s'assurer que l'interface est intuitive et facile à utiliser, tout en répondant aux besoins de l'utilisateur. Pour notre simulateur d'incendie de forêt, nous avons utilisé l'outil *Figma* pour concevoir l'interface graphique. Cela nous a permis de créer une représentation visuelle de l'application avant même de commencer à coder, ce qui nous a aidé à identifier les fonctionnalités clés et à organiser l'interface de manière logique. La figure 3.7 présente le design préalable que nous avons réalisé sur *Figma* tandis que la figure 3.8 présente l'interface effectif qui a été implémenté.



Figure 3.7 – Design Figma de l'interface graphique

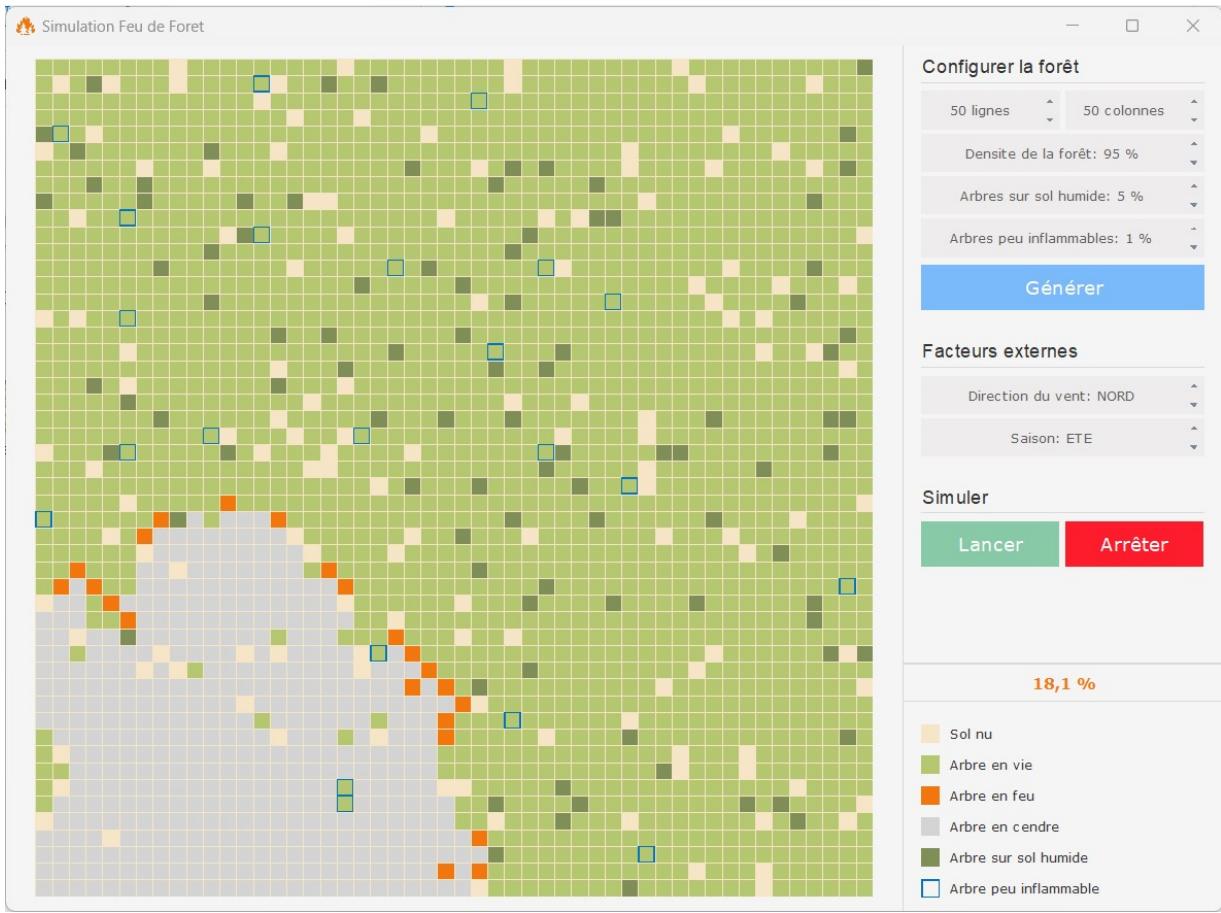


Figure 3.8 – Interface graphique réellement implémenté

L’interface graphique est composée de deux parties principales : la grille qui représente la forêt à gauche, et le menu de configuration à droite.

La grille représente la forêt et est subdivisée en cellules, chacune représentant un arbre ou un sol nu. L’utilisateur peut modifier l’état d’une cellule en cliquant dessus plusieurs fois. Les arbres vivants peuvent également être modifiés en faisant un clic droit pour indiquer s’ils se trouvent sur un sol humide et/ou s’ils sont inflammables.

Le menu de configuration est subdivisé en quatre parties principales. La première partie permet à l’utilisateur de configurer la forêt en indiquant le nombre de lignes et de colonnes, la densité de la forêt, le taux d’arbres sur sol humide et le taux d’arbres peu inflammables souhaité dans la forêt. Un bouton générer permet d’appliquer cette configuration sur la grille.

La deuxième partie permet de configurer les facteurs externes tels que la direction du vent et la saison. Ces paramètres peuvent influencer la propagation de l’incendie et permettent aux utilisateurs de voir comment les changements externes peuvent affecter la simulation.

La troisième partie contient deux boutons qui permettent de lancer et d'arrêter la simulation. Ces boutons sont facilement accessibles pour que l'utilisateur puisse commencer ou mettre en pause la simulation rapidement.

La quatrième et dernière partie contient le pourcentage de forêt déjà brûlée ainsi qu'une légende expliquant les différentes couleurs dans la grille. Cette partie permet à l'utilisateur de suivre l'état de la simulation et de comprendre ce que chaque couleur représente dans la grille.

En résumé, le design de l'interface graphique est une étape importante dans le développement de tout logiciel, car il permet de concevoir visuellement la façon dont les utilisateurs vont interagir avec l'application. Dans le cas de notre simulateur de feu de forêt, l'interface graphique est composée d'une grille représentant la forêt et d'un menu de configuration permettant à l'utilisateur de modifier différents paramètres. La conception claire et logique de l'interface permet à l'utilisateur de comprendre rapidement les différentes fonctionnalités et de les utiliser efficacement.

3.5 Mise en œuvre de la solution à l'aide du langage Java

Cette partie présente trois algorithmes essentiels pour le bon fonctionnement de la simulation. Parmi ceux-ci, nous retrouvons l'algorithme de la boucle de simulation qui permet d'itérer sur chaque cellule de la grille afin de déterminer l'état futur de celle-ci. De plus, l'algorithme de calcul de l'état futur d'une cellule et l'algorithme de calcul de la probabilité de propagation du feu sur une cellule sont également des algorithmes clés dans notre simulateur. Dans cette partie, nous allons décrire de manière détaillée ces algorithmes pour mieux comprendre leur fonctionnement et leur rôle dans la simulation.

3.5.1 Algorithme de la boucle de simulation

L'algorithme de la boucle de simulation de la figure 3.9 est l'un des éléments clés de la solution. Cet algorithme est responsable de parcourir toutes les cellules de la grille et de calculer leur état futur en fonction de la direction du vent et de la saison. La méthode renvoie un tableau de deux entiers, le premier étant 0 si la simulation doit s'arrêter (s'il n'y a plus d'arbres en feu) et 1 sinon. Le deuxième entier représente le pourcentage de la forêt déjà brûlée.

La boucle de simulation commence par initialiser trois variables : continuerSimulation, nombreTotalArbre et nombreArbreEnCendre. La variable continuerSimulation est initialement à 0, mais elle sera mise à 1 si une cellule en feu est trouvée. Les variables nombreTotalArbre et nombreArbreEnCendre sont utilisées pour calculer le pourcentage de la forêt déjà brûlée. La boucle itère ensuite sur toutes les cellules de la grille. Pour chaque cellule, la méthode calculeEtatFutur est appelée, ce qui calcule l'état futur de la cellule en fonction de la direction du vent et de la saison. Si l'état futur de la cellule est 2 (en feu), la variable continuerSimulation est mise à 1. Enfin, les variables nombreTotalArbre et nombreArbreEnCendre sont mises à jour en fonction de l'état actuel et futur de la cellule. Le pourcentage de la forêt déjà brûlée est calculé en divisant le nombre d'arbres

```

public double[] simulation(String direction_vent, String saison){
    double continuerSimulation = 0;
    double nombreTotalArbre = 0;
    double nombreArbreEnCendre = 0;

    for(int i = 0; i < this.nb_lignes; i++) {
        for( int j = 0; j < this.nb_colonnes; j++) {
            Cellule cellule = this.getCellule(i, j);
            cellule.calculeEtatFutur(direction_vent, saison);
            if(cellule.getEtatFutur() == 2) continuerSimulation =
                1;
            if(cellule.getEtat() == 1 || cellule.getEtat() == 2
                || cellule.getEtat() == 3) nombreTotalArbre += 1;
            if(cellule.getEtat() == 3 || cellule.getEtatFutur()
                == 3) nombreArbreEnCendre += 1;
        }
    }

    return new double[]{continuerSimulation, ((nombreArbreEnCendre/nombreTotalArbre)*100)};
}

```

Figure 3.9 – Code de la méthode Grille.simulation()

en cendres par le nombre total d’arbres et en multipliant par 100. Le tableau de deux entiers est ensuite retourné.

L’algorithme de la boucle de simulation est un élément essentiel de la solution car il permet de simuler la propagation du feu dans la forêt en calculant l’état futur de chaque cellule. Cet algorithme, ainsi que les autres algorithmes utilisés dans la solution, ont été implémentés en Java.

3.5.2 Algorithme de calcul de l’état futur d’une cellule

Cette méthode visible sur la figure 3.10 utilise plusieurs critères pour déterminer l’état futur d’une cellule : le voisinage, la direction du vent, la saison, l’humidité du sol et le type de végétation.

Tout d’abord, si la cellule est un arbre déjà en cendre, son état futur reste le même. Si la cellule est un arbre en feu, son état futur est défini comme étant en cendre. Si la cellule est un arbre vivant et qu’aucun de ses voisins n’est en feu, elle reste dans son état actuel. Cependant, si la cellule est un arbre vivant et qu’au moins un de ses voisins est en feu, un calcul de probabilité est effectué pour déterminer si le feu va se propager à cette cellule. La probabilité de propagation du feu vers la cellule est calculée à l’aide de la méthode `calculeP()` qui prend en compte la direction du vent, la saison, l’humidité du sol et le type de végétation. Cette probabilité est ensuite utilisée pour faire passer le cas

```

@Override
public void calculeEtatFutur(String direction_vent, String saison
){
    // p : probabilité de propagation du feu vers la cellule
    double p;

    if(this.etat == 3) this.etat_futur = etat;
    else if(this.etat == 2) this.etat_futur = 3;
    else if(this.etat == 1){
        if(this.nombreDeVoisinsEnFeu() == 0) this.etat_futur =
            this.etat;
        else{
            p = this.calculeP(direction_vent, saison);
            if(this.feuVaSePropager(p)) this.etat_futur = 2;
            else this.etat_futur = etat;
        }
    }
}

```

Figure 3.10 – Code de la méthode Arbre.calculeEtatFutur()

échéant la cellule à l'état en feu.

3.5.3 Algorithme de calcul de la probabilité de propagation

L'algorithme de calcul de la probabilité de propagation du feu présenté sur la figure 3.11 est relativement simple. La méthode prend en entrée la direction du vent et la saison courante et calcule la probabilité que l'arbre prenne feu en se basant sur plusieurs critères.

Tout d'abord, la méthode prend en compte l'impact du voisinage sur la probabilité p . Si plusieurs cellules voisines sont en feu, alors la probabilité que l'arbre prenne feu sera plus grande. Ainsi, pour chaque voisin en feu, la probabilité p est augmentée de 0.05. Ensuite, la méthode prend en compte l'impact de la direction du vent sur la probabilité p . Si le vent souffle dans la direction de la cellule, la probabilité que l'arbre prenne feu sera plus grande. Ainsi, si le vent souffle dans la direction de la cellule, la probabilité p est augmentée de 0.5. Si le vent souffle dans une autre direction, la probabilité p est diminuée de 0.1.

La méthode prend également en compte l'impact de la saison sur la probabilité p . Si la saison est l'été, alors la probabilité que l'arbre prenne feu sera plus grande. Ainsi, si la saison est l'été, la probabilité p est augmentée de 0.2. Si la saison est l'hiver, la probabilité p est diminuée de 0.1. Ensuite, la méthode prend en compte l'impact de l'humidité du sol sur la probabilité p . Si le sol est humide, alors la probabilité que l'arbre prenne feu sera plus faible. Ainsi, si le sol est humide, la probabilité p est diminuée de 0.1. Enfin, la méthode prend en compte l'impact du type de végétation sur la probabilité p . Si l'arbre est peu inflammable, alors la probabilité que l'arbre prenne feu sera plus faible. Ainsi, si

```

public double calculeP(String direction_vent, String saison){
    // Impact du voisinage sur p.
    double p = 0.6 + this.nombreDeVoisinsEnFeu() * 0.05;
    // Impact de la direction du vent sur p.
    if(!this.feuVersCellule(direction_vent) && direction_vent != null) p -= 0.1;
    if(this.feuVersCellule(direction_vent)) p += 0.5;
    // Impact de la saison sur p
    if(saison == null){}
    else if(saison.equals("HIVER")) p -= 0.1;
    else if(saison.equals("ETE")) p += 0.2;
    // Impact de l'humidité du sol sur p.
    if(this.solEstHumide()) p -= 0.1;
    // Impact du type de végétation sur p.
    if(this.arbreEstPeuInflammable()) p -= 0.2;
    // La probabilité doit être comprise en 0 et 1.
    if(p < 0) p = 0;
    if(p > 1) p = 1;

    return p;
}

```

Figure 3.11 – Code de la méthode Arbre.calculeP()

l’arbre est peu inflammable, la probabilité p est diminuée de 0.2.

En sortie, la méthode retourne la probabilité p, qui est comprise entre 0 et 1.

Conclusion

En conclusion, ce projet de simulation de feu de forêt a permis de mettre en place un modèle simple mais réaliste de propagation du feu, en se basant sur plusieurs critères tels que la direction du vent, la saison, l'humidité du sol et le type de végétation. Le modèle a été implémenté en utilisant le langage de programmation Java et en se basant sur le principe de la programmation orientée objet.

Nous avons vu comment l'algorithme de calcul de l'état futur d'une cellule ainsi que celui de la probabilité de propagation du feu ont été développés. Ces deux algorithmes sont les pierres angulaires de la simulation de feu de forêt et sont essentiels pour comprendre comment le feu se propage et comment il peut être maîtrisé.

Il est important de noter que d'autres facteurs peuvent également influencer la propagation du feu de forêt, tels que la topographie, la densité de la forêt, la distance entre les arbres, la présence d'obstacles tels que des rivières ou des routes, et bien d'autres encore. Ces facteurs peuvent être intégrés dans la simulation pour en augmenter la précision.

En outre, il serait possible d'intégrer des techniques d'intelligence artificielle pour prédire la direction du vent en se basant sur les données des feux de forêt passés. Cela pourrait permettre de mieux prévoir la propagation du feu et d'adopter des mesures de prévention plus efficaces.

Enfin, ce projet de simulation de feu de forêt est un exemple de la façon dont la programmation peut être utilisée pour simuler des phénomènes naturels complexes et pour aider à mieux comprendre les mécanismes qui les sous-tendent. Il est donc possible d'explorer d'autres domaines en utilisant cette approche et de créer des modèles de simulation pour comprendre des phénomènes encore plus complexes.

Bibliographie

- [1] L. HERNÁNDEZ ENCINAS et al. « Modelling forest fire spread using hexagonal cellular automata ». In : *Applied Mathematical Modelling* 31.6 (2007), p. 1213-1227. DOI : 10.1016/j.apm.2006.04.001. URL : <https://doi.org/10.1016/j.apm.2006.04.001>.
- [2] I. KARAFYLLIDIS et A. THANAILAKIS. « A model for predicting forest fire spreading using cellular automata ». In : *Ecological modeling* 99.1 (1997), p. 87-97. DOI : 10.1016/s0304-3800(96)01942-4. URL : [https://doi.org/10.1016/s0304-3800\(96\)01942-4](https://doi.org/10.1016/s0304-3800(96)01942-4).
- [3] A. ALEXANDRIDIS et al. « A cellular automata model for forest fire spread prediction : The case of the wildfire that swept through Spetses Island in 1990 ». In : *Applied Mathematics and Computationg* 204.1 (2008), p. 191-201. DOI : 10.1016/j.amc.2008.06.046. URL : <https://doi.org/10.1016/j.amc.2008.06.046>.
- [4] S. G. BERJAK et J. W. HEARNE. « An improved cellular automaton model for simulating fire in a spatially heterogeneous Savanna system ». In : *Ecological Modelling* 148.2 (2002), p. 133-151. DOI : 10.1016/s0304-3800(01)00423-9. URL : [https://doi.org/10.1016/s0304-3800\(01\)00423-9](https://doi.org/10.1016/s0304-3800(01)00423-9).
- [5] B. DROSSEL et F. SCHWABL. « Self-Organized Critical Forest-Fire Model ». In : *Physical Review Letters* 69.11 (1992), p. 1629-1632. DOI : 10.1103/physrevlett.69.1629. URL : <https://doi.org/10.1103/physrevlett.69.1629>.
- [6] T. GHISU et al. « An optimal Cellular Automata algorithm for simulating wildfire spread ». In : *Environmental Modelling Software* 71 (2015), p. 1-14. DOI : 10.1016/j.envsoft.2015.05.001. URL : <https://doi.org/10.1016/j.envsoft.2015.05.001>.
- [7] K. MUTTHULAKSHMI et al. « Simulating forest fire spread and fire-fighting using cellular automata ». In : *Chinese Journal of Physicse* 65 (2020), p. 642-650. DOI : 10.1016/j.cjph.2020.04.001. URL : <https://doi.org/10.1016/j.cjph.2020.04.001>.
- [8] J. G. FREIRE et C. C. DACAMARA. « Using cellular automata to simulate wildfire propagation and to assist in fire management ». In : *Natural Hazards and Earth System Sciences* 19.1 (2019), p. 169-179. DOI : 10.5194/nhess-19-169-2019. URL : <https://doi.org/10.5194/nhess-19-169-2019>.
- [9] A. HERNÁNDEZ ENCINAS et al. « Simulation of forest fire fronts using cellular automata ». In : *Advances in Engineering Software* 38.6 (2007), p. 372-378. DOI : 10.1016/j.advengsoft.2006.09.002. URL : <https://doi.org/10.1016/j.advengsoft.2006.09.002>.
- [10] Andrew WUENSCHÉ et Michael LESSER. *Cellular Automata : A Discrete View of the World*. John Wiley & Sons, 2000.
- [11] G. FRANCESCHINI et et AL. « Cellular Automata Models of Fire Spread ». In : *International Journal of Wildland Fire* (2020).

- [12] Henry Laurence GANTT. *Work, Wages and Profits*. Engineering Magazine Co., 1916.
- [13] R. V. LESIKAR et M. E. FLATLEY. *Basic Business Communication : Skills For Empowering the Internet Generation*. McGraw Hill, 2001.
- [14] OBJECT MANAGEMENT GROUP. *Unified Modeling Language™ (UML®) - OMG*. consulté en 2023. URL : <https://www.omg.org/spec/UML/2.5.1/> (visité le 15/05/2023).
- [15] S. CHACON et B. STRAUB. *Pro Git*. 2nd ed. Apress, 2014.
- [16] *Project git repository*. 2023. URL : https://github.com/Gladys11-cloud/Simulation_Feu_de_Foret (visité le 29/05/2023).

A Autre diagramme de classe proposé par l'équipe 1

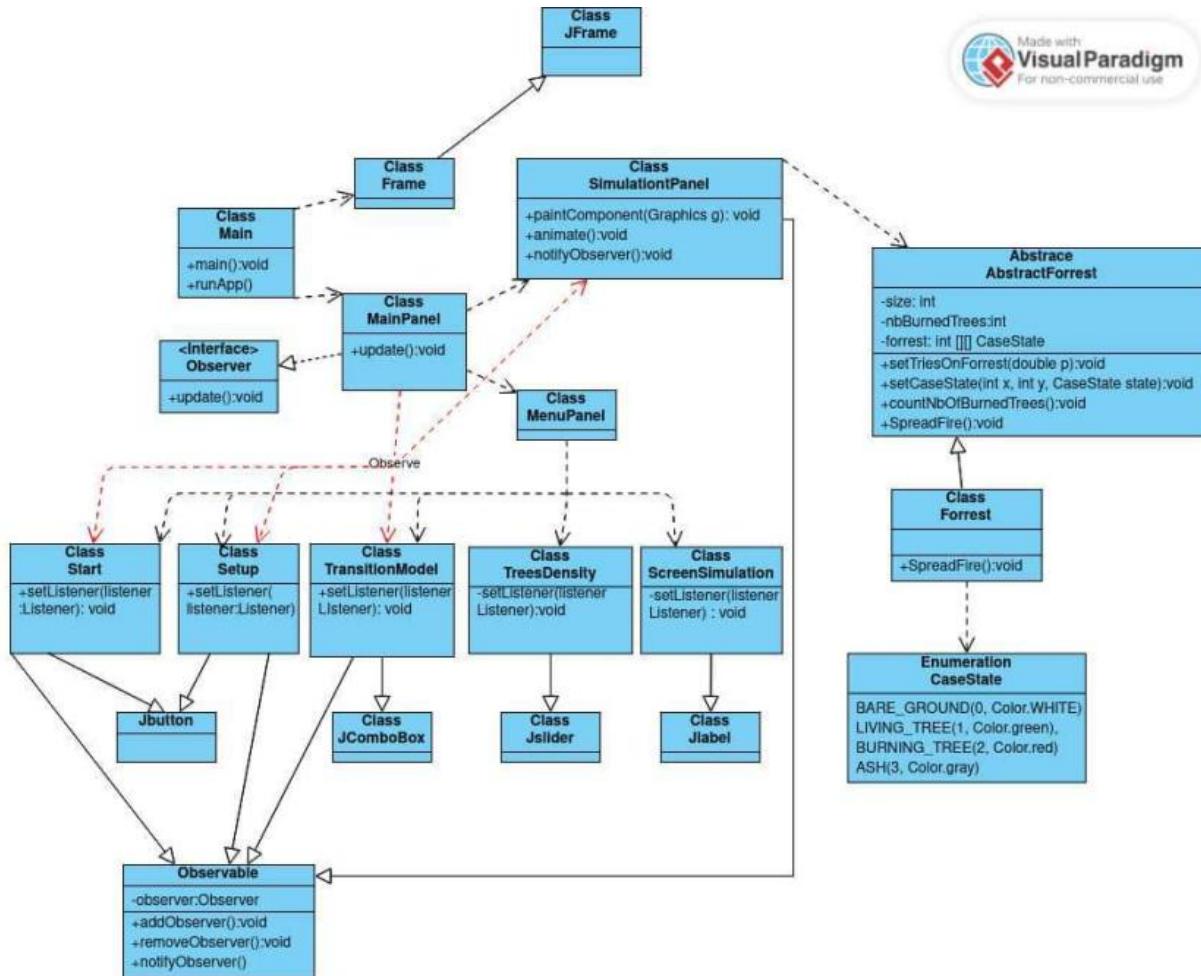


Figure A.1 – Version finale du diagramme de classes

B Autre diagramme de classe proposé par l'équipe 2

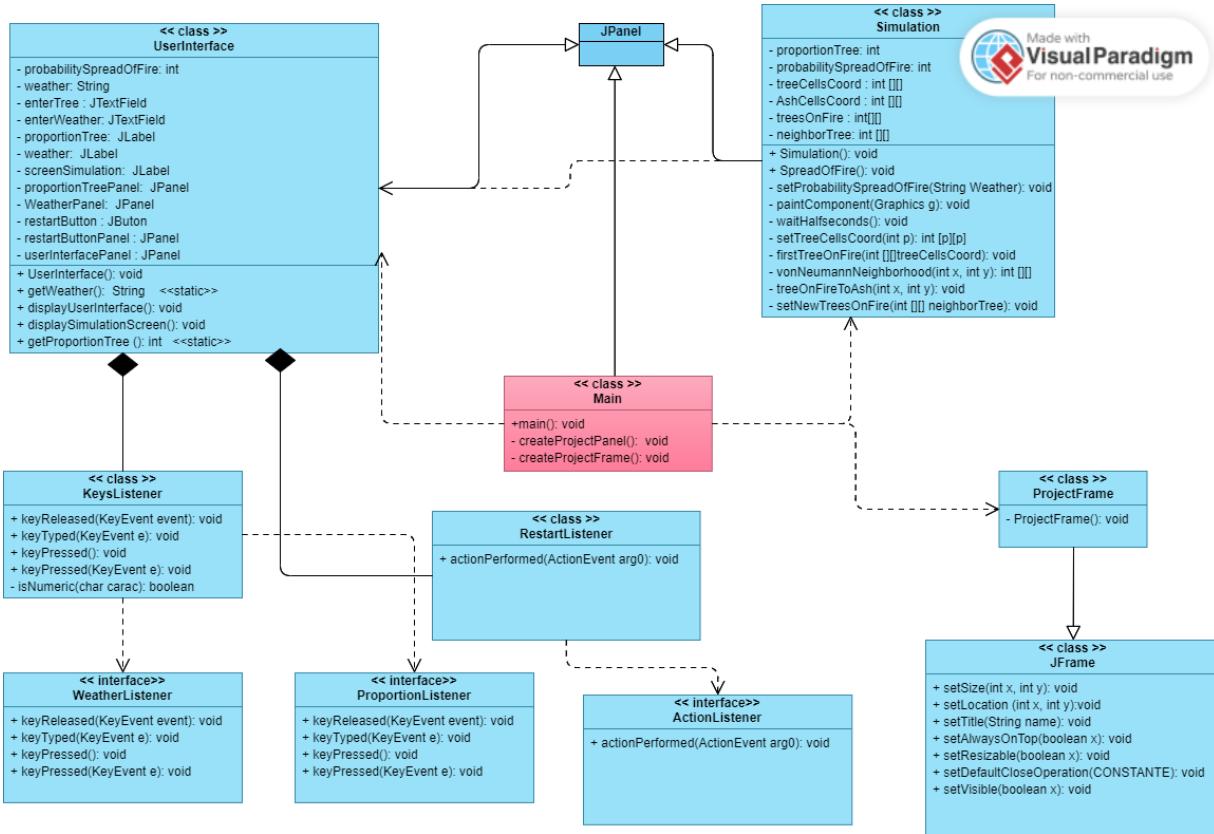


Figure B.1 – Autre diagramme de classe proposé par l'équipe 2