Université de Paris 2020/2021

« Corona Bounce »

Projet de

Aliche Zahra, Bragina natalia, Golikova Anna, SaHI KENZA, RAVENEAU EMILIEN

[Introduction 3](#_Toc70461268)

[Cahier des charges 3](#_Toc70461269)

[Les objets principaux 3](#_Toc70461270)

[Réglages possibles 3](#_Toc70461271)

[Coté GUI 4](#_Toc70461272)

[Schéma des classes 5](#_Toc70461273)

[Méthodologie et déroulement du projet 7](#_Toc70461274)

[Réalisation de contamination 7](#_Toc70461275)

[Réalisation de scenarios 7](#_Toc70461276)

[Réalisation de Model–View–Controller 8](#_Toc70461277)

[Gestion des threads 8](#_Toc70461278)

[Réalisation de la partie graphique 9](#_Toc70461279)

[Difficultés rencontrées 10](#_Toc70461280)

[Problèmes connus 10](#_Toc70461281)

[Pistes d’extensions 10](#_Toc70461282)

# Introduction

Pour notre UE de projet de notre deuxième année de Licence en informatique à l’université de Paris Diderot, nous avons travaillé sur un projet de Corona Bounce. Au cours de ces trois mois de durs labeurs, nous avons utilisé nos connaissances déjà acquises au cours de nos parcours respectifs, ainsi que de nouvelles apprises au cours de ce délai, pour offrir un simulateur d’épidémie. Ce projet est réalisé en groupe, dans lequel chacun de nous a apporté sa pierre à l’édifice. Il est constitué de cinq étudiants de parcours informatique/math info :

- ALICHE ZAHRA

- BRAGINA NATALIA

- GOLIKOVA ANNA

- SAHI KENZA

- RAVENEAU EMILIEN

Le projet réalisé est proposé par nos encadrants : Mr Yan Jurski, et sous la direction de Mr Aldric Degorre, Responsable des projets du département informatique de l’Université.

# Problématique :

Développer un simulateur d’épidémie qui permet l’observation de sa propagation dans des populations virtuelles en variant les mesures et les paramètres, en analysant les courbes de contamination et de guérison, de faire un point sur l’efficacité des stratégies adoptées et de déduire les facteurs inhibiteurs et stimulateurs de la diffusion de la maladie.

.

# Cahier des charges

L’application offre la possibilité de lancer l’animation avec les paramètres de base, les changer à partir de GUI et de voir les conséquences.

## Les objets principaux

Les points qui représentes les individus, qui bougent et qui changent son couleur selon son état (seins, contaminés/contagieux, malades et guéris).

Les graphes affichent en temps réel le développement de l’épidémie en parallèle avec le nombre de personnes malades/contagieuses, saines et guéries.

En appuyant sur les boutons du menu principal on peut :

1. Lancer le mouvement (« Start »)
2. Arrêter le mouvement et le reprendre (« Pause/Resume »)
3. Changer les réglages (« Settings »)
4. Recharger les populations à nouveau tout en gardant les réglages ainsi que scenarios (« Reset »)
5. Montrer la légende pour chaque élément de la scène (« ? »).

## Réglages possibles

Notre programme permet à l’utilisateur d’effectuer les changements tant les paramètres de populations et de maladie, que de choisir les scenarios appliques par gouvernement.

Manettes de réglages à partir de « Settings » permettent varier les valeurs de paramètres suivantes :

1. Maladie

* Durée de l’incubation.
* Durée de recouvrement.
* Durée de l’immunité.
* Rayon de contamination.

1. Population

* Nombre de personnes.

1. Espace

* Nombre de murs/frontières (qui restreignent le mouvement des points).
* Vitesse de fermeture les frontières.

Outre cela nous proposons de choisir parmi les scenarios suivants :

* « No scénario » (déplacement aléatoire et non limité).
* « Soft Lockdown » (déplacement s’effectue dans une zone de quelque rayon de position initial d’individu).
* « Boundaries » (déplacement aléatoire mais limité par les frontières).

« Strict Lockdown » (une grande partie de population est confinée, qu’une petite partie de population se déplace librement – médecines, pompiers etc.).

* « Soft Lockdown + Boundaries » (application deux scenarios à la fois).

## Coté GUI

Les fenêtres et ses éléments s’adaptent aux dimensions de l’écran.

Il y a une légende expliquant les objets et les fonctionnalités impliqués.

Les fenêtres sont redimensionnables. Toutes les parties des fenêtres ainsi que les objets de la simulation s’adaptent aux changement de dimensions dynamiquement.

# Schéma des classes

Diagramme des classes du modèle :

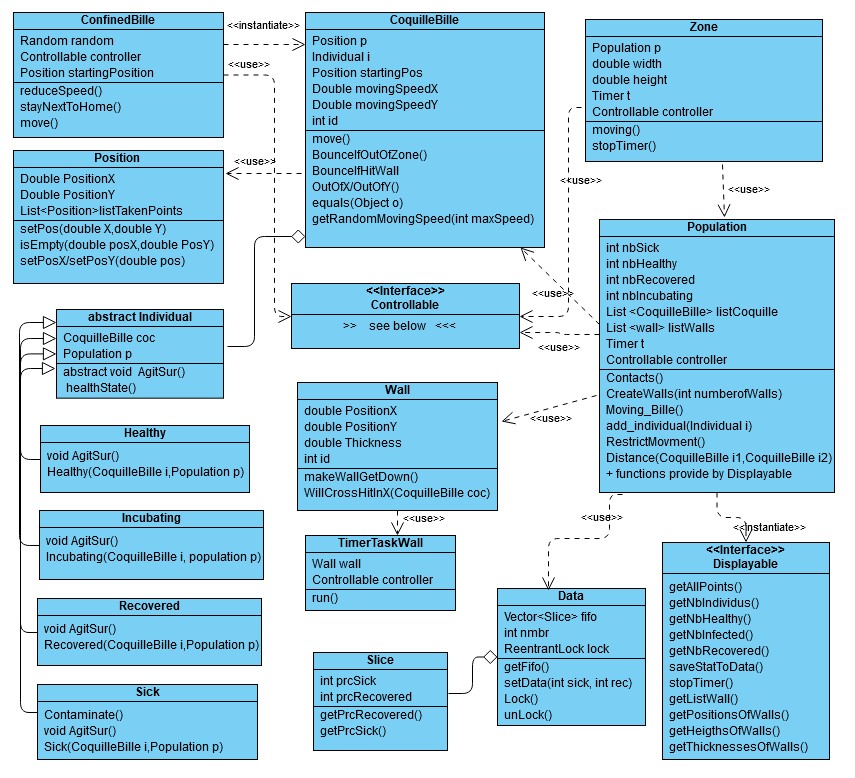


Diagramme des classes du vue et controller :



Diagramme de main classes : 

# Méthodologie et déroulement du projet

Nous avons réalisé presque tout ce que nous avions défini initialement dans le cahier des charges.

## Réalisation de contamination

La **Zone** représente l’environnement de vie d’une population. Elle est caractérisée par une hauteur et largeur. Nous avons une population ayant un certain nombre d’individus, ils circulent dans un environnement Zone. Les individus sont perçus comme des points de la Zone d’étude.

Les individus représentés par la classe **Individual**, peuvent prendre 4 états : sain, malade, contagieux et guéri. Il y a 4 classes représentant ces 4 états de santé possible de chaque individu : **Sick**, **Healthy**, **Recovered** et **Incubating**.

Capacités de l’Individu de se déplacer sont réalisé par la classe **CoquilleBille**, qui encapsule un individu. CoquilleBille a une positionu et ne vitesse 2-dimensionnelle.

**ConfinedBille** est un type spécial de CoquilleBille encapsulant les individus d’une population soumise à un confinement (SoftLockdown).

Ils ont une vitesse bornée par une valeur maximale et circulent aléatoirement (déplacements libres) si aucun scénario n’est appliqué, de manière restreinte dans le cas contraire.

Les objets **Wall** sont des frontières restreignant les déplacements des individus (déplacements régionaux uniquement). Chaque frontière a une position X et Y et une largeur. Dans l’implémentation, les frontières sont descendantes.

Au début de la simulation, la population est dispersée aléatoirement sur la surface de la zone. Par la suite, les points circulent avec une certaine vitesse aléatoire bornée par une valeur maximale (que l’on peut changer). La contamination se produit si un point sain est à un rayon\_de\_contamination près d’un point malade/contagieux. Il rentre dans une phase d’incubation (avant l’apparition des symptômes) avant qu’il ne soit compté parmi les malades. Il devient guéri après une période de temps et acquiert une immunité qui le protège contre une nouvelle infection temporairement.

## Réalisation de scenarios

**SoftLockDown** : Dans ce cas de figure, un type spécifique de CoquilleBille(s) est créé : ConfinedBille(s) caractérisé par une vitesse et un rayon de déplacement restreints. Les individus ne se déplacent que dans un voisinage de leur position initiale et avec une vitesse réduite.

**StrictLockdown** : Une grande partie de la population est forcée de rester chez elle, aucun déplacement n’est toléré. Une minorité de gens peuvent circuler, cela concerne notamment ceux qui exercent des métiers indispensables au bon fonctionnement et à la sécurité de la population, policiers, docteurs, pompiers, livreurs, ... Ou aux urgences.

**Boundaries** : Des frontières divisent progressivement la zone en sous-zones pour limiter les déplacements (déplacements régionaux uniquement).

Progressivement ? Les frontières ne sont pas placés instantanément mais petit à petit le long de la zone. Les points sont bornés par ces dernières et ne peuvent changer de région.

**Boundaries + Softlockdown** : Une combinaison entre frontières et confinement cités plus haut.

Les statistiques montrent bien que le taux de contamination diminue conséquemment.

## Réalisation de Model–View–Controller

Notre projet est organisé selon MVC design pattern. Pour effectuer la connexion entre Model, Vue et Controller nous utilisons deux interfaces :

* **Displayable** qui sert les fonctions nécessaires à Model pour bien être affiché dans Vue les points et les murs, dessiner les graphs et gérer les timers.  
  Alors, Displayable consiste les getters de liste de points, de nombres de seins, malades, contagieux, guéris, ainsi que les getters de murs, ses positions et sa hauteur, les fonctions servent à sauvegarder la statistique en historique et fermer les tâches de timer.
* **Controllable** qui sert les fonctions nécessaires à transmettre au modèle les paramètres prises à partir de GUI. Donc ce sont les getters et les setters da l’espace, de population, de maladie, des murs et de scenarios, ainsi que les fonctions pour changer l’état de threads.

Nous avons créé la classe **Controller**, dont rôle consiste à être auxiliaire au Main Conroller et Settings Controller.

Controller contient les valeurs initiales des paramètres pour lancer le modèle. En outre, en tant qu’il implémente l’interface Controllable, il offre les fonctions nécessaires pour changer ces valeurs initiales à partir de UI (user interface) pendant l’usage de l’application.

## Gestion des threads

Dans notre projet nous avons implémenté le multi-thread architecture.

Etant donné on a 5 threads : l’un qui est principale (celui de App.java) et deux pour chaque une des populations (un pour gérer le déplacement des points, l’autre pour gérer les cycles de maladie et sauvegarder l’historique).

A part de main thread, les threads se créent dans deux classes : classe Population et classe Zone.

Le thread de timer crée dans la Population partagent les classes suivantes :

* Classe Incubating – pour gérer la durée d’incubation de maladie, temps écoulé - Incubating devient Sick. Durée = durée d’incubation.
* Classe Sick – pour gérer la durée de maladie après le contact – dès que le temps s’écoule l’individu Sick change la classe pour Recovered. Durée = durée de maladie.
* Classe Recovered – pour gérer la durée de non contamination, temps écoulé - recovered devient healthy. Durée = durée d’immunité.
* Classe Population – pour sauvegarder la statistique afin de la représenter en graph.
* Classe Mur – pour fermer les frontières.

Le thread de timer crée dans la Zone serve à effectuer le déplacement des individus.

L’animation dans GUI est attachée au thread principal.

Afin d’assurer la synchronisation des processus, on met à pause tous les timers à partir de création des instances. Pour cela on utilise une variable enum (« Paused » au début) qui est passée par l’interface Cotrollable dans toutes les classes concernées (Zone, Population, Sick, Incubating, Recovered, Wall). Ce n’est qu’appuyant sur le bouton « Start » que cette variable change sa valeur pour « Working » et tous les processus commencent à tourner.

On assure la synchronisation des données grâce à la classe Reentrantlock.

Il est utilisé :

* pour changer la valeur de l’enum de l’état de threads (interface Controllable et classe Controller)
* dans le cycle de dessin de graphs afin de synchroniser les processus de lecture et d’écriture de statistiques (classe Data).

Reentrantlock offre la plus souple fonctionnalité que la synchronisation implicite offerte par synchronized. Ainsi Reentrantlock nous permet de séparer les processus de lock en unlock et les implémenter dans les fonctions différentes. Cela nous permet d’assurer l’accès unique à partir de thread principale à l’objet de Data pendant le dessin les graphs et procéder la synchronisation après. Lecture et écriture de enum state, qui s’effectue à partir de plusieurs méthodes et plusieurs threads, sont aussi parfaitement contrôlées avec Reentrantlock.

## Réalisation de la partie graphique

La partie graphique est réalisée à l’aide du format fxml. Deux fichiers fxml, corona bounce.fxml et settings.fxml correspondent aux deux fenêtres du programme, c’est-à-dire la fenêtre principale où la simulation se passe et la fenêtre des paramètres.

Le format fxml nous a permis de regrouper les éléments essentiels de la partie graphique. Pour ce faire nous avons utilisé l’application SceneBuilder et nous avons aussi modifié le code directement.

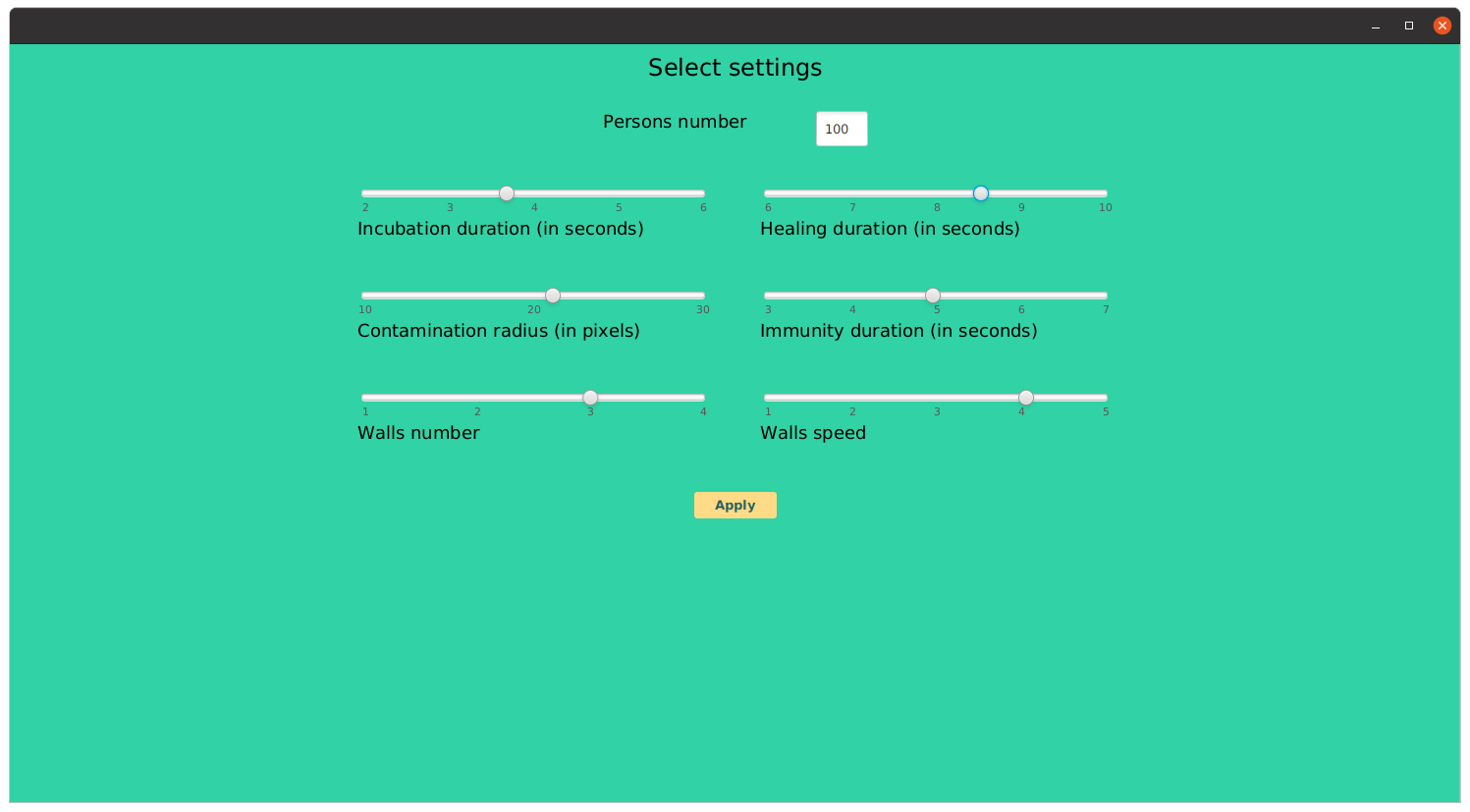
La création des éléments graphiques est possible aussi bien dans fxml que dans les parties .java ; nous nous sommes servis de ce fait pour dessiner les graphes. En effet, on a déplacé la création des graphs en java car cela nous permet de les appliquer les styles à l’aide de .css plus facilement.

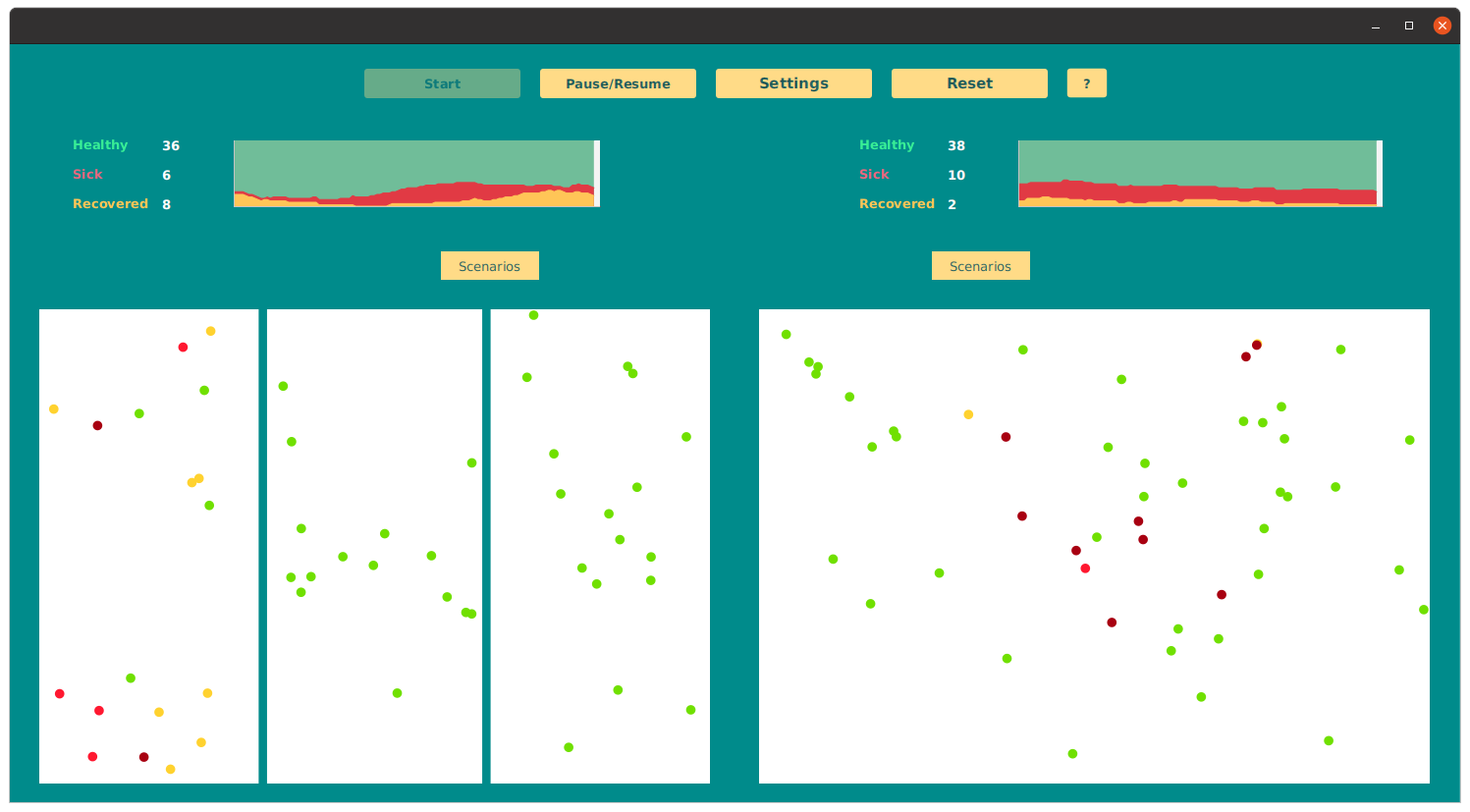
La création des graphes se fait dans la classe Graph qui est usé par MainController. Le dessin les frontières et les points/individus se fait par la classe Renderer.

On définit certains styles dans style.css ; pourtant une partie des caractéristiques de style se fait dans les fichiers fxml directement (surtout dans les cas où les éléments sont difficilement accessibles via css).

Pour la création des fenêtres, on utilise les attributs essentiels d’une application JavaFX, à savoir Scene et Stage que nous définissons dans App.java.







JavaFX

Maven

JavaDoc

# Difficultés rencontrées

# Problèmes connus

Paramétrer Intellij à chaque fois qu’il y a un commit de pom (ou autre de ce type) par hasard…

# Pistes d’extensions

* Frontières fermée, ouverture s’effectue après.
* Réglage de la vitesse des points.

# Conclusion :

Nous avons fait preuve d’une patience et d’une camaraderie qui ont été des atouts dans l’élaboration de notre projet. La répartition du travail et la présence de chacun à toutes les réunions ont permis un développement dans de très bonnes conditions.