

Rapport de projet : Corona-Bounce

-groupe corona 4 :

ABDELLATIF Kenzi : k.abdellatif03@gmail.com

SIMONNET William : william.mpsim@gmail.com

VERHAEGHE Jean-Yves : thevanderer@gmail.com

HANNAH Destiny : hannadestiny@icloud.com

Sommaire :

1)Définition du projet.....	2
a)Enoncé.....	2
b)Plan.....	2
c)Présentation générale.....	2
2)Cahier des charges.....	3
3)Structure.....	4
a)Structure du projet.....	4
b)Schéma des classes.....	6
4)Outils et Langages.....	7
5)Exemples du codes.....	8
a)Fenêtre de menu.....	8
b)Édition d'environnement.....	9
c)Lancement d'une simulation.....	11
d)La simulation.....	12
e)Fin de la simulation.....	15
6)Difficultés rencontrées.....	17
7)Problèmes connus.....	18
8)Extensions possibles.....	18
9)Documentation.....	18
10)Détails d'exécution.....	19

1). Définition du projet

a. Enoncé

Il s'agit d'un simulateur d'épidémie de coronavirus en java. Le but de ce projet est de mettre en évidence les différentes mécaniques liées à l'épidémie et l'influence de certains paramètres sur celle-ci.

b. Plan

Nous présentons dans ce document les différentes démarches que nous avons suivi pour réaliser le projet. En particulier le système de création d'environnement et le modèle épidémique. Pour cela, on utilisera un diagramme des classes ainsi que des lignes de codes tirées du projet. On parlera aussi du déroulement du projet et de son organisation.

c. Présentation générale

La simulation commence par une fenêtre de menu où on a le choix entre trois possibilités.

- Éditer l'environnement
- Lancer la simulation
- Regarder les notes sauvegardées

Dans l'éditeur, on peut ajouter des rectangles et des ellipses, les supprimer et les déplacer dans l'environnement. Il y a un bouton pour sauvegarder la disposition courante et un autre pour revenir vers le menu.

En lançant la simulation, on a d'abord une fenêtre nous demandant les détails de la simulations. Après les avoir remplis, on peut confiner, limiter les déplacements aériens et mettre sur pause. On peut alors voir sur la droite des graphiques montrant l'évolution de l'épidémie.

Un des boutons permet aussi d'arrêter la simulation et déplace vers une fenêtre de notes où l'utilisateur peut transcrire ses remarques et les points qu'il juge important à prendre en compte.

Libre à l'utilisateur de sauvegarder ou non ces notes auxquelles il pourra accéder à partir du menu.

2. Cahier des charges

Voici les fonctionnalités offertes par notre programme :

Confinement : Permettre à l'utilisateur de confiner toute la population dans des emplacements prédéfinis.

Soins : La simulation doit avoir un mécanisme de soin des personnes contaminées ou en état critique en temps réel.

Restaurant : La simulation doit recréer un lieu de type fermé où les personnes deviennent plus ou moins statiques.

Aéroport : La simulation doit fournir un système de contamination extérieur à travers l'aéroport.

Cimetière : La simulation offre un endroit où se placeraient les personnes décédées.

Collisions : Les personnes qui se croisent ou croisent un obstacle changent de direction

Contaminations : Après une collision ou un contact en proximité, Il y a une certaine probabilité de contamination d'une personne saine si l'autre personne est contaminée.

Interface de paramétrage : Permettre à l'utilisateur de choisir certains paramètres liés à la simulation.

Gestion de la maladie en temps réel : La situation de la personne contaminée évolue en temps réel.

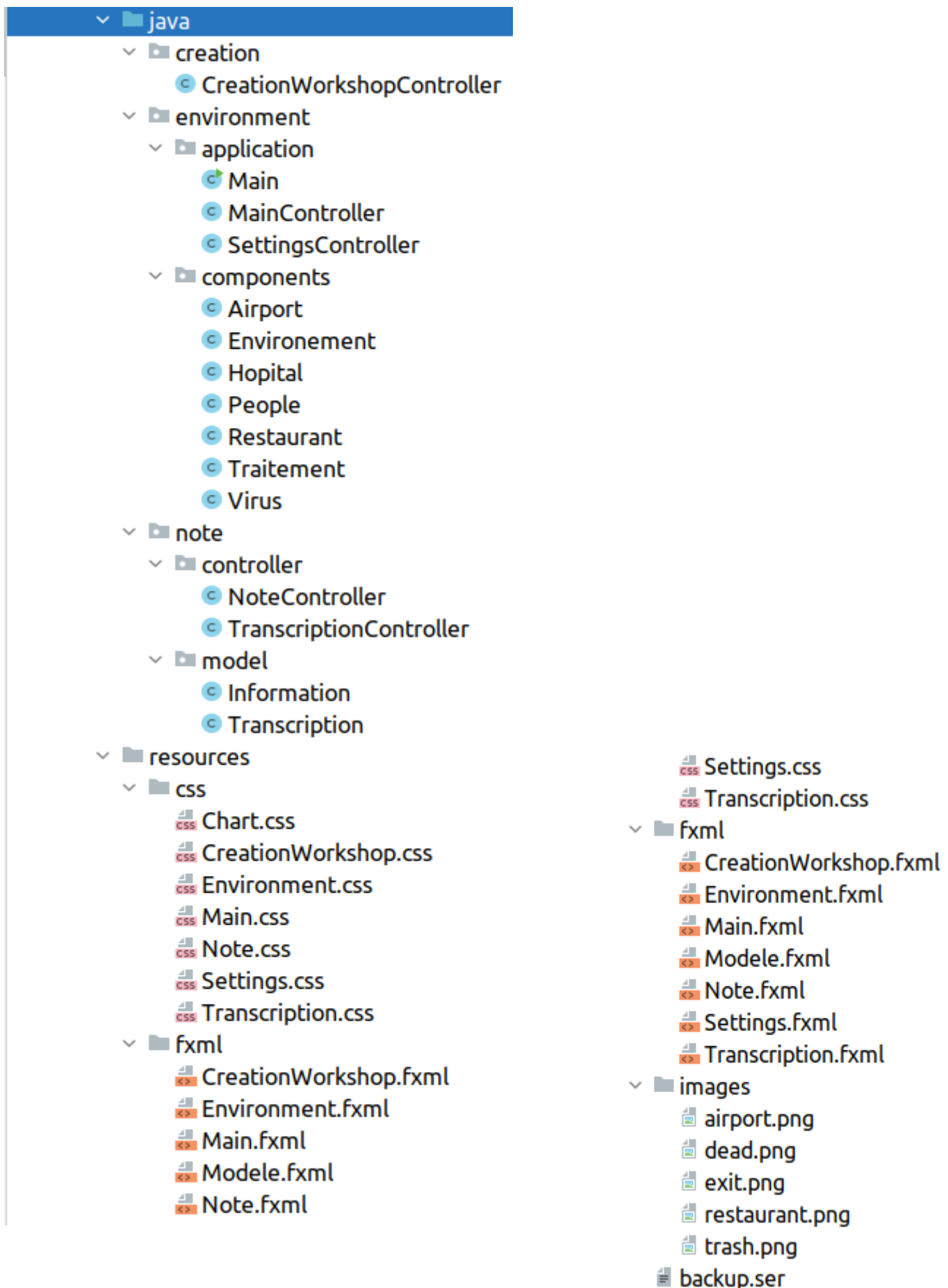
Graphiques : Présence de courbes montrant l'évolution de la situation en temps réel.

Editeur d'environnement : Mise à disposition de l'utilisateur d'une interface lui permettant de créer lui même son environnement. Cet éditeur doit être facile à utiliser et à comprendre.

Gestion des notes : Permettre à l'utilisateur d'interrompre temporairement la simulation pour sauvegarder des notes concernant la simulation actuelle.

3. Structure

a. Structure du projet



Pour organiser le projet de sorte à avoir une structure bien définie, on a implémenté le modèle MVC (Model View Controller).

Java → **creation** : Edition de l'environnement

→ **environnement** :

→ **application** : les classes de lancement de l'application et du réglage pré-simulation.

→ **components** : Composants de l'environnement de simulation.

→ **note** :

→ **controller** : Interaction avec l'interface graphique qui gère l'édition de notes de fin de simulation

→ **model** : modèle de note transcrite par l'utilisateur (objet qui sera sauvegardé)

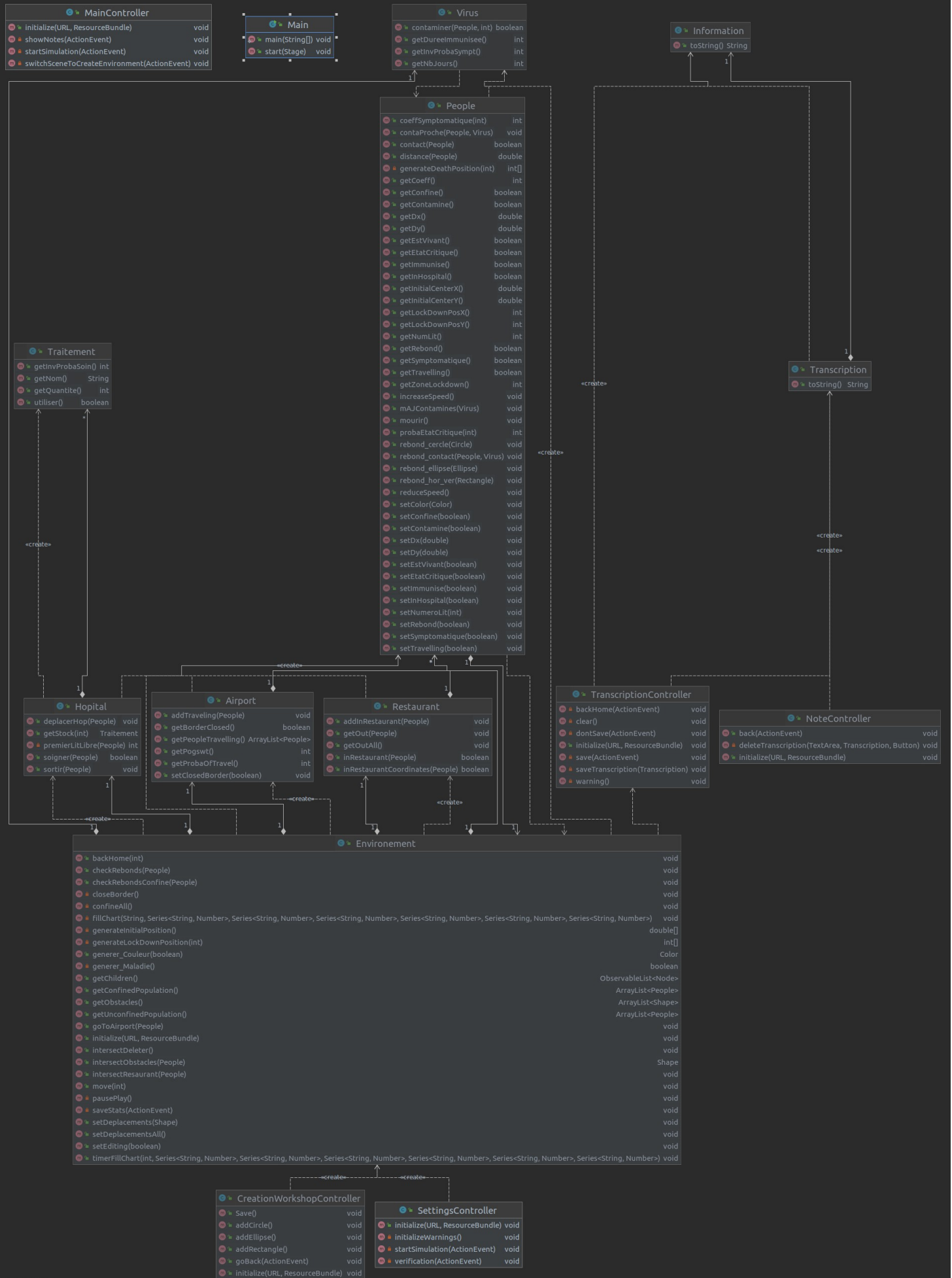
Ressources → **css** : feuilles de style utiles à la « Vue » du programme.

→ **fxml** : Vue du programme.

→ **images** : images incluses dans la « Vue » du projet.

→ backup.ser : fichier de sauvegarde des objets de note.

b. Schéma des classes



4.Outils et Langages

a. Javafx

Nous avons choisi javafx comme bibliothèque graphique car elle représentait une bibliothèque riche et performante plus configurable et plus souple que ses semblables.

b. Gradle

L'utilisation de gradle facilitait la compilation et l'exécution du projet et a été décidée suite à son utilisation durant le projet de groupe (prépro2) du premier semestre. Il a été préférable d'utiliser un outil que nous maîtrisions ou du moins connaissions déjà.

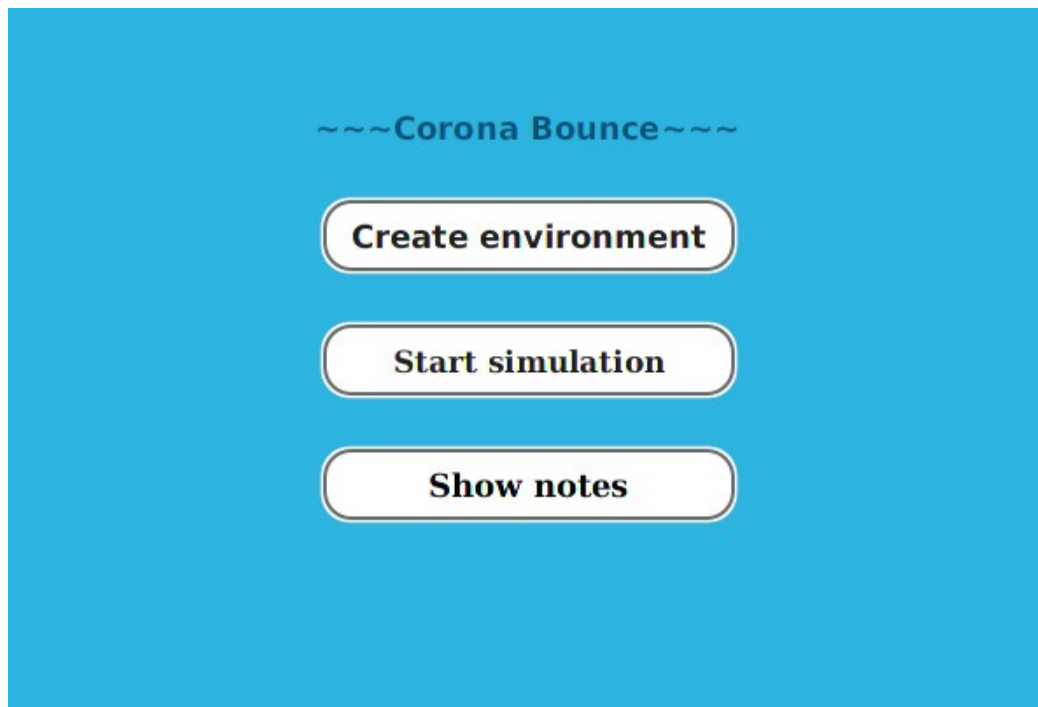
c. fxml

L'utilisation de fichiers fxml pour gérer la partie « Vue » du projet s'est montrée à nos yeux indispensable et s'est révélée être un gain de temps énorme.

5. Exemples de codes

Nous ne donnerons ici que des extraits intéressants de notre code.

a.Fenêtre de menu



Au lancement du programme, c'est la première fenêtre affichée.

```
@Override
public void start(Stage primaryStage) throws IOException {
    primaryStage.setTitle("Corona-Bounce !");

    String location = "../resources/fxml/Main.fxml";
    FXMLLoader loader = new FXMLLoader();
    loader.setLocation(getClass().getResource(location));
    Parent root = loader.load();

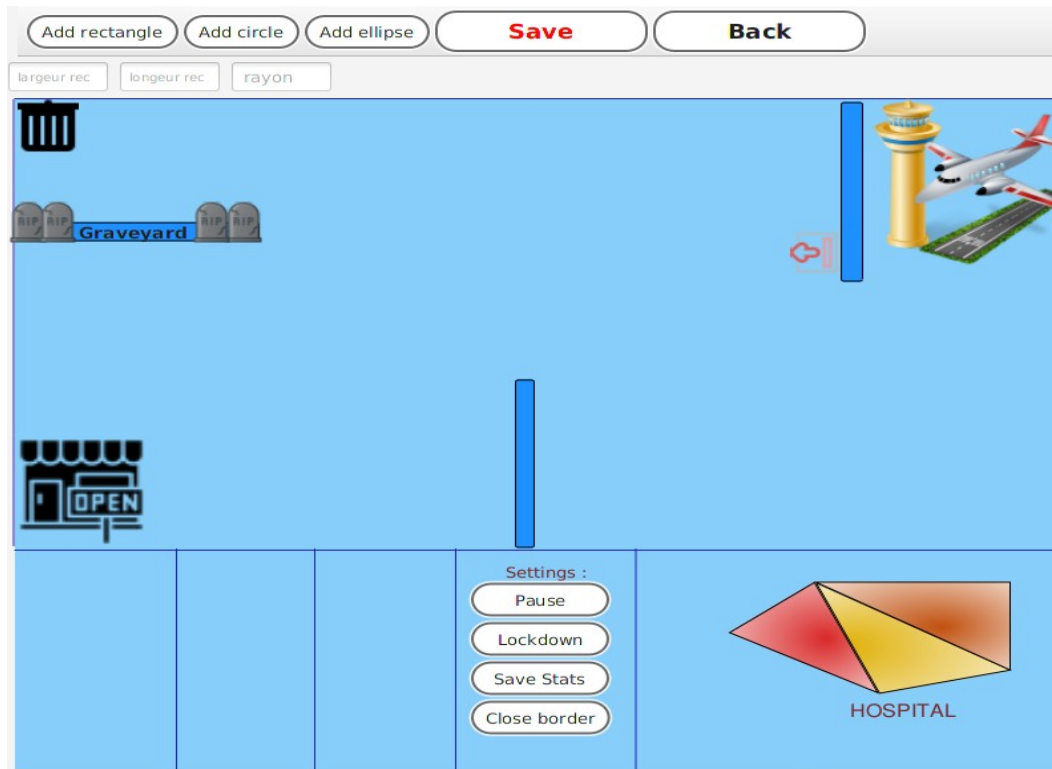
    Scene scene = new Scene(root, 600, 400);

    primaryStage.setScene(scene);
    primaryStage.setResizable(false);
    primaryStage.show();
}

public static void main(String[] args) {
    Application.launch(args);
}
```


La fonction start consiste à charger le décor du programme et initialiser la taille de la fenêtre. Celle-ci est gérée par la classe mainController.

b. Édition d'environnement



Cette fenêtre s'affiche après que l'utilisateur ait appuyé sur le bouton « Create environment » de la fenêtre précédente grâce au code suivant :

```
@FXML
private void switchSceneToCreateEnvironment(ActionEvent e){
    try {

        String location = "../../resources/fxml/CreationWorkshop.fxml";
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource(location));
        Parent creationWorkshopParent = loader.load();

        Scene scene = new Scene(creationWorkshopParent);

        Stage window = (Stage)((Node)e.getSource()).getScene().getWindow();

        window.setScene(scene);
        window.setResizable(false);
        window.show();

    } catch (Exception exception){
        exception.printStackTrace();
    }
}
```

Après que les modifications sont finies, le bouton « save » permet de sauvegarder grâce à la fonction « save » qui se trouve dans la classe « CreationWorkshopController ».

Voici par exemple un rectangle sauvegardé dans la fonction « save » :

```
ObstaclesFXML.add(  
    "  
        <Rectangle fx:id=\"obstacle\" + idfx + \" \" +  
            \" \" arcHeight=\"5.0\" \" +  
            \"arcWidth=\"5.0\" \" +  
            \"fill=\"DODGERBLUE\" \" +  
            \"height=\"\" + h + \" \" +  
            \"layoutX=\"\" + lX + \" \" +  
            \"layoutY=\"\" + lY + \" \" +  
            \"stroke=\"BLACK\" \" +  
            \"strokeType=\"INSIDE\" \" +  
            \"width=\"\" + w + \" \" />  
    );
```

(Les paramètres étant données par l'utilisateur)

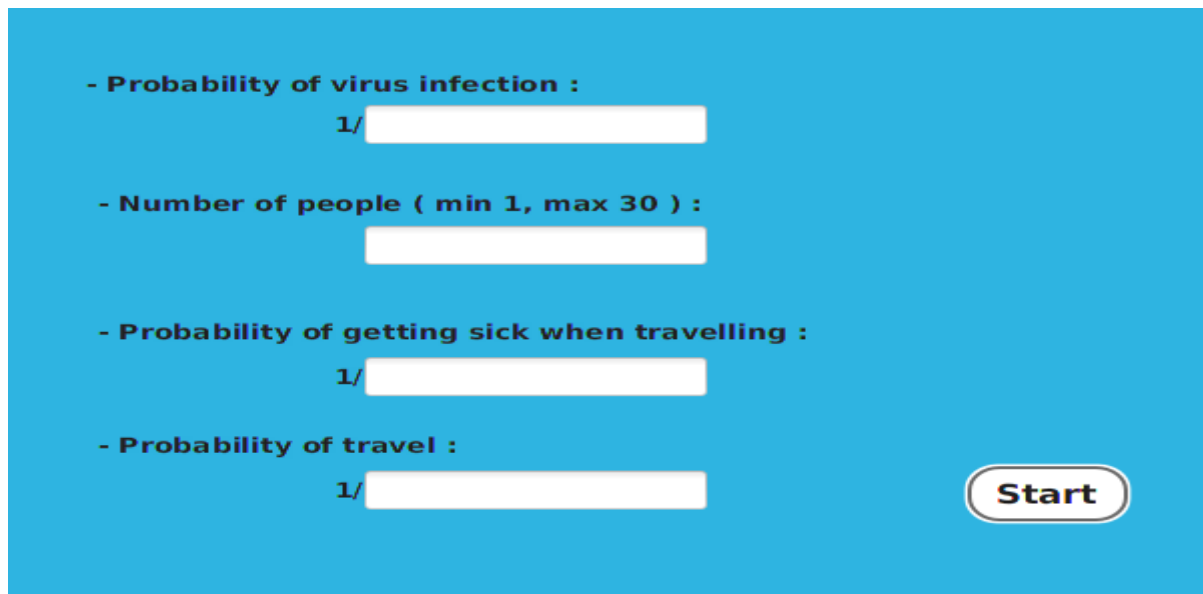
Les autres boutons parlent d'eux même.

Voici par exemple le code associé au bouton « back » :

```
public void goBack(ActionEvent e) {  
    try {  
        String location = "../resources/fxml/Main.fxml";  
        FXMLLoader loader = new FXMLLoader();  
        loader.setLocation(getClass().getResource(location));  
        Parent root = loader.load();  
        Scene scene = new Scene(root, 600, 400);  
        Stage stage = (Stage)((Node)e.getSource()).getScene().getWindow();  
        stage.setTitle("Corona-Bounce");  
        stage.setScene(scene);  
        stage.setResizable(false);  
        stage.show();  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
}
```

c. Lancement d'une simulation

Tout d'abord, on peut voir la fenêtre de configuration s'afficher après que l'utilisateur cherche à commencer une simulation.



- Probability of virus infection :
1/

- Number of people (min 1, max 30) :

- Probability of getting sick when travelling :
1/

- Probability of travel :
1/

Start

Après avoir rempli les paramètres ci-dessus, l'appui du bouton « start » renvoie l'utilisateur vers le début de la simulation grâce à la fonction suivante :

```
private void startSimulation(ActionEvent e){
    try {
        Environement root = new Environement(this.nbrPeopleInt,
        this.probaOfGettingSickWhenTravelInt,
        this.probaOfVirusInfectionInt, this.probaOfTravelInt);

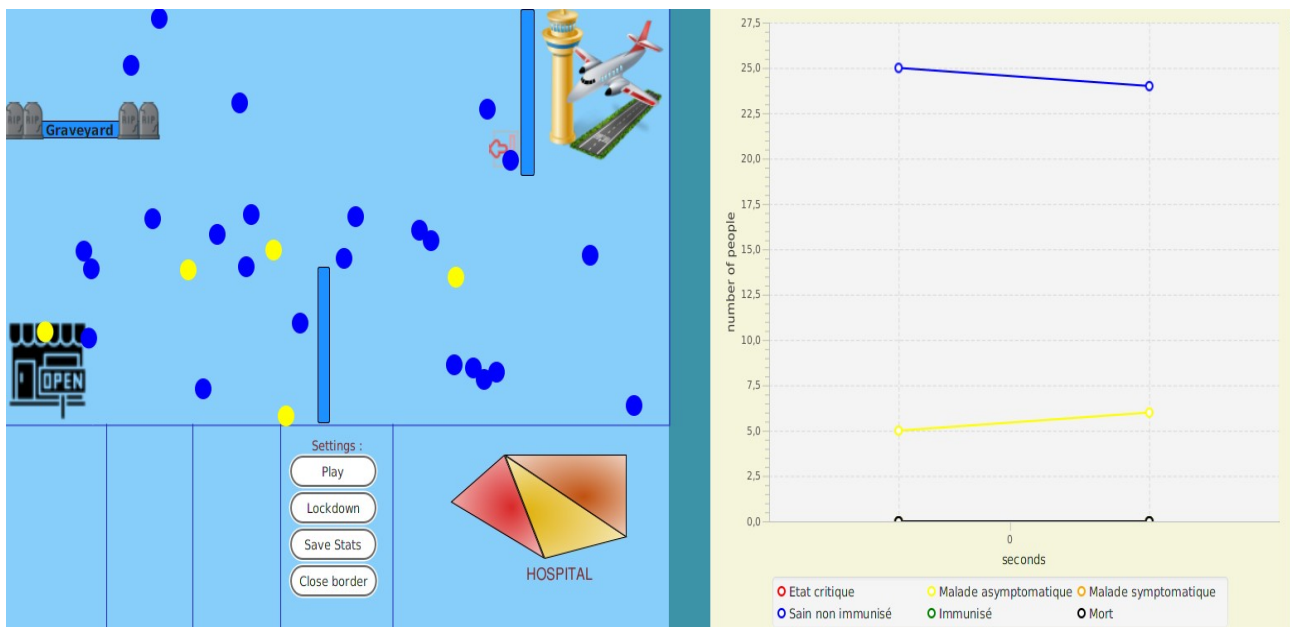
        Scene scene = new Scene(root);

        Stage window = (Stage)((Node)e.getSource()).getScene().getWindow();
        root.timerFillChart(1000, root.LineChart.getData().get(0)
        , root.LineChart.getData().get(1), root.LineChart.getData().get(2),
        root.LineChart.getData().get(3), root.LineChart.getData().get(4),
        root.LineChart.getData().get(5));
        root.move(50);
        root.backHome(3000);
        window.setScene(scene);
        window.setResizable(false);
        window.show();
    } catch (Exception exception){
        exception.printStackTrace();
    }
}
```

Dans cette fonction, la partie centrale charge les graphiques et actualise leurs données chaque seconde.

La fonction « move » utilisée dans cette fonction regroupe l'implémentation des différents aspects de l'évolution de la simulation, chaque 50ms.

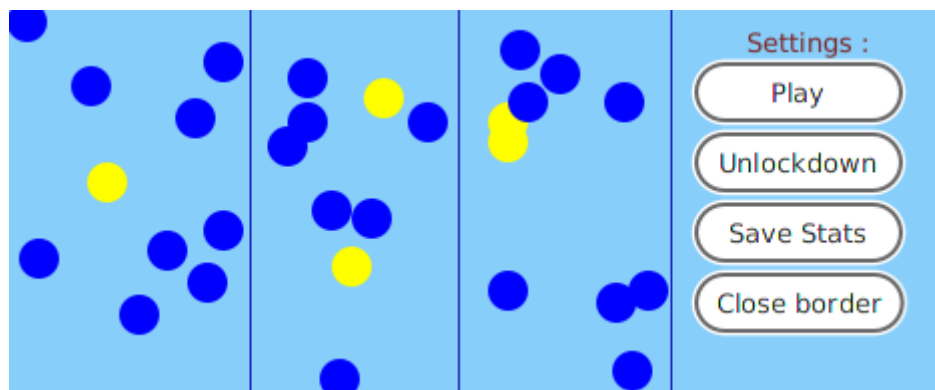
d. La simulation



→ **boutons :**

pause/play : permet de figer la simulation.

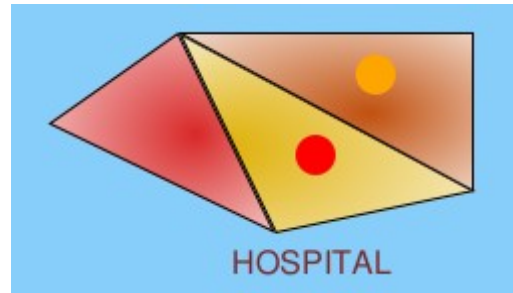
Lockdown : modélise le confinement et le déconfinement



« Save stats » permet d'ouvrir une fenêtre pour écrire des remarques qui seront sauvegardées et accessibles après la simulation.

« Close border » permet d'arrêter les voyages.

→ **L'hôpital :**

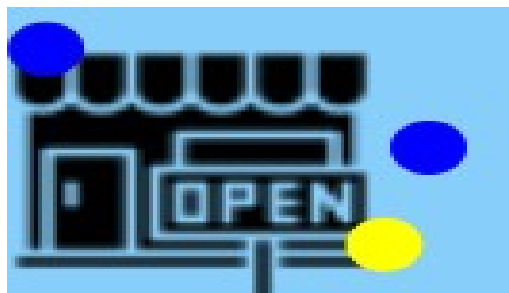


Les personnes peuvent être soit saines non immunisées (bleues), soit asymptomatiques (jaunes), soit symptomatiques (orange), soit en état critique (rouges) soit mortes (noires).

Dans le cas où une personne devient symptomatique, elle peut être amenée à l'hôpital. Sur les trois lits, l'un d'entre eux est cependant réservé aux états critiques.

Une fois à l'hôpital, ils sont soignés avec un certain traitement dépendant de leur état (critique ou non).

→ **Le restaurant**



Zone où ralentissent les personnes simulant un endroit clos de rassemblement.

```
public void addInRestaurant(People p){  
    if ( this.inRestaurantCoordinates(p) ) {  
        this.peopleInRestaurant.add(p);  
        p.reduceSpeed();  
    }  
}
```

```
public void getOut(People p){  
    if ( !this.inRestaurantCoordinates(p) ){  
        this.peopleInRestaurant.remove(p);  
        p.increaseSpeed();  
    }  
}
```

→ L'aéroport



Il est définie par des capteurs qui vérifient si une personne entre en contact avec les bords de l'aéroport .

Si oui, la personne qui est entrée en contact a une certaine probabilité de voyager. Puis elle disparaîtra pendant une certaine durée limitée et réapparaîtra à la sortie de l'aéroport avec une certaine probabilité d'être contaminée.

→ **Le cimetière :**



C'est une zone où les mort (boules noires) sont rassemblés, ils n'interagissent alors plus avec l'environnement.

e. Fin de la simulation

Après avoir appuyé sur le bouton « Save Stats », la simulation se retrouvera figée et une fenêtre additionnelle apparaîtra permettant la transcription et la sauvegarde de données ou de remarques.

Voici à quoi cela ressemble :

Collect information

Personal note Chart note Important points Save

Welcome : the fields noted "*To be completed" must be completed before saving

Name of the experience : *To be completed

Note : *To be completed

Collect information

Personal note Chart note Important points Save

About :

-Sick people : *To be completed

-Unaffected people : *To be completed

-Sick people without symptoms : *To be completed

-People healed : *To be completed

Collect information

Personal note Chart note Important points Save

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.

Collect information

Personal note Chart note Important points Save

Don't save

Save


```

private void saveTranscription(Transcription t){
    Transcription[] transcriptions;
    try {
        FileInputStream fis = new FileInputStream("src/main/resources/backup.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        try {
            Transcription[] tab = ( Transcription[] ) ois.readObject() ;
            transcriptions = new Transcription[ tab.length + 1 ];
            System.arraycopy(tab, 0, transcriptions, 0, tab.length);
        } catch ( Exception e ) {
            transcriptions = new Transcription[1];
        }
    } catch (Exception e ) {
        transcriptions = new Transcription[1];
    }
    try {
        FileOutputStream fos = new FileOutputStream("src/main/resources/backup.ser");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        transcriptions[ transcriptions.length - 1 ] = t;
        oos.writeObject( transcriptions );
        oos.close();
    } catch ( Exception e ) {
        e.printStackTrace();
    }
}

```

Après la transcription de l'information, l'utilisateur aura le choix de sauvegarder ou non ses notes. La sauvegarde se fait alors grâce à la fonction suivante :

6). Difficultés rencontrées

Il a fallut d'abord beaucoup se documenter sur javafx, fxml et gradle.

Ensuite, il a été difficile de trouver la bonne manière d'implémenter le fait de déplacer les obstacles dans l'éditeur.

Pour la collision, il y avait eu un problème car certaines fois les boules étaient en collision permanentes et restaient donc « attachées ».

Pour régler le problème il a fallut ajouter un boolean «rebond » qui déterminait si la balle pouvait ou non encore rebondir.

Pour les rebonds, il avait été aussi envisagé de faire des rebonds plus précis (en calculant l'angle etc.), mais cela semblait trop lent à exécuter et certaines balles se traversaient sans collision, on a donc abandonné cette piste et fait des rebonds plus simples.

Changer la couleur des graphes pour qu'ils correspondent à ce qu'ils représentent a aussi été sur la fin du projet un problème jusqu'à ce qu'on ait l'idée d'utiliser du css pour le résoudre.

7). Problèmes connus

Les fonctions de rebonds sur les bords de l'environnement ainsi que les fonctions de génération d'emplacements de confinement sont codées en dur, ce qui n'est pas forcément la meilleure manière de faire.

8). Extensions possibles

On comptait faire un système multi-environnement où plusieurs environnements pourraient interagir entre eux.

Il y avait aussi la possibilité d'étoffer encore le modèle en faisant vieillir la population. En effet, après la première vague, on peut voir que l'épidémie ne se relance pas car les personnes sensibles sont déjà mortes. (Et dans ce cas pourquoi pas aussi un système de naissance)

Il y avait aussi comme possibilité le réapprovisionnement de l'hôpital durant la simulation et pourquoi pas un compteur montrant le coût de l'épidémie en temps réel.

Enfin, il aurait été possible aussi d'ajouter d'autres graphes montrant l'impact de l'épidémie selon les tranches d'âge comme c'était un critère important concernant le virus.

9).Documentation

Le code est fourni avec une documentation : le répertoire « JavaDoc »

Ce répertoire JavaDoc contient des fichiers html de cette dite documentation.

Le fichier index.html est à exécuter en premier dans un navigateur web pour ensuite pouvoir naviguer librement dans une documentation agréable à voir.

10). Détails d'exécution

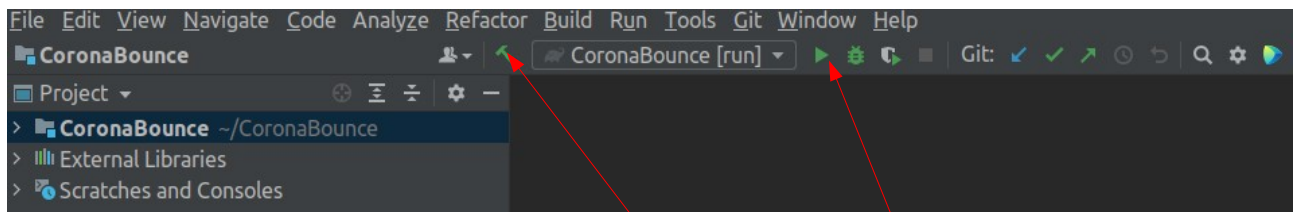
a) Exécution via le terminal

```
kenzi@ubuntu:~$ cd CoronaBounce/  
kenzi@ubuntu:~/CoronaBounce$ ./gradlew build  
  
> Configure project :  
Project : => no module-info.java found  
  
Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.  
Use '--warning-mode all' to show the individual deprecation warnings.  
See https://docs.gradle.org/6.7/userguide/command_line_interface.html#sec:command_line_warnings  
  
BUILD SUCCESSFUL in 1s  
6 actionable tasks: 5 executed, 1 up-to-date  
kenzi@ubuntu:~/CoronaBounce$ ./gradlew run _
```

→ Accéder au répertoire du projet qui aura été cloné à partir de gitlab.

→ Executer ./gradlew build et ./gradlew run dans l'ordre.

b) Via IntelliJ



1)

2)

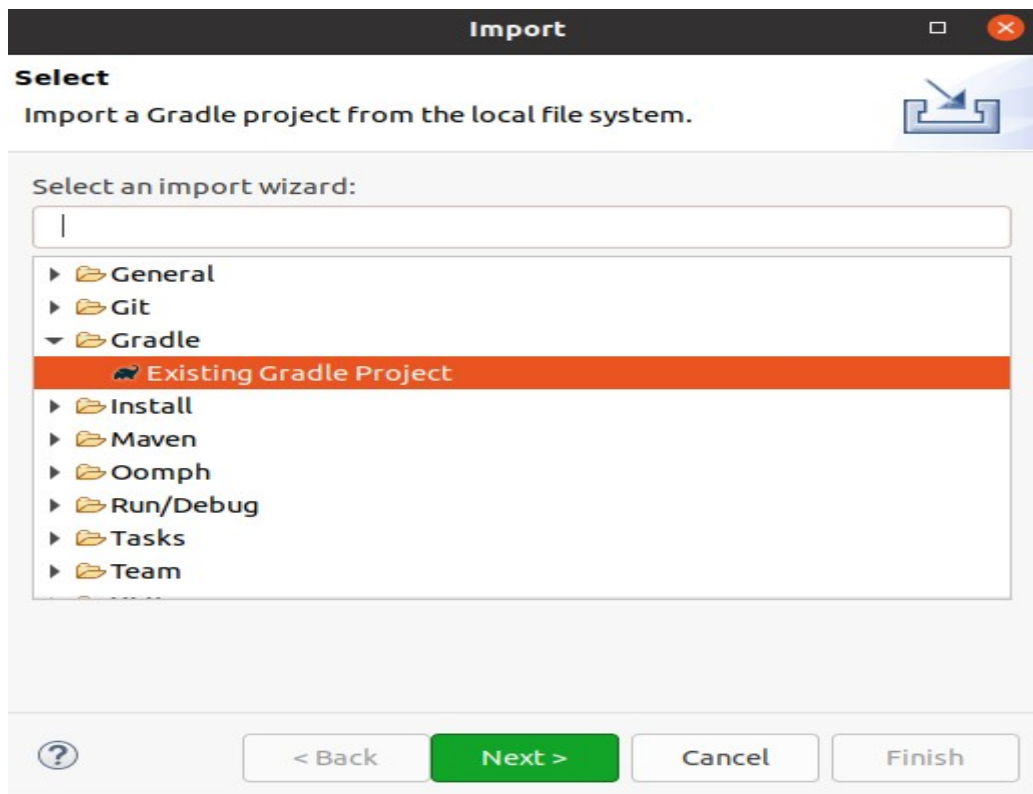
Après l'ouverture du répertoire du projet via IntelliJ comme un projet gradle :

1) Cliquez sur le marteau pour build le projet

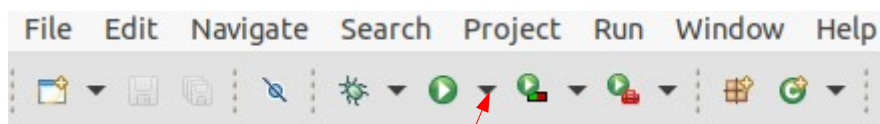
2) Cliquez sur l'icône pour exécuter le projet

c)Via Eclipse

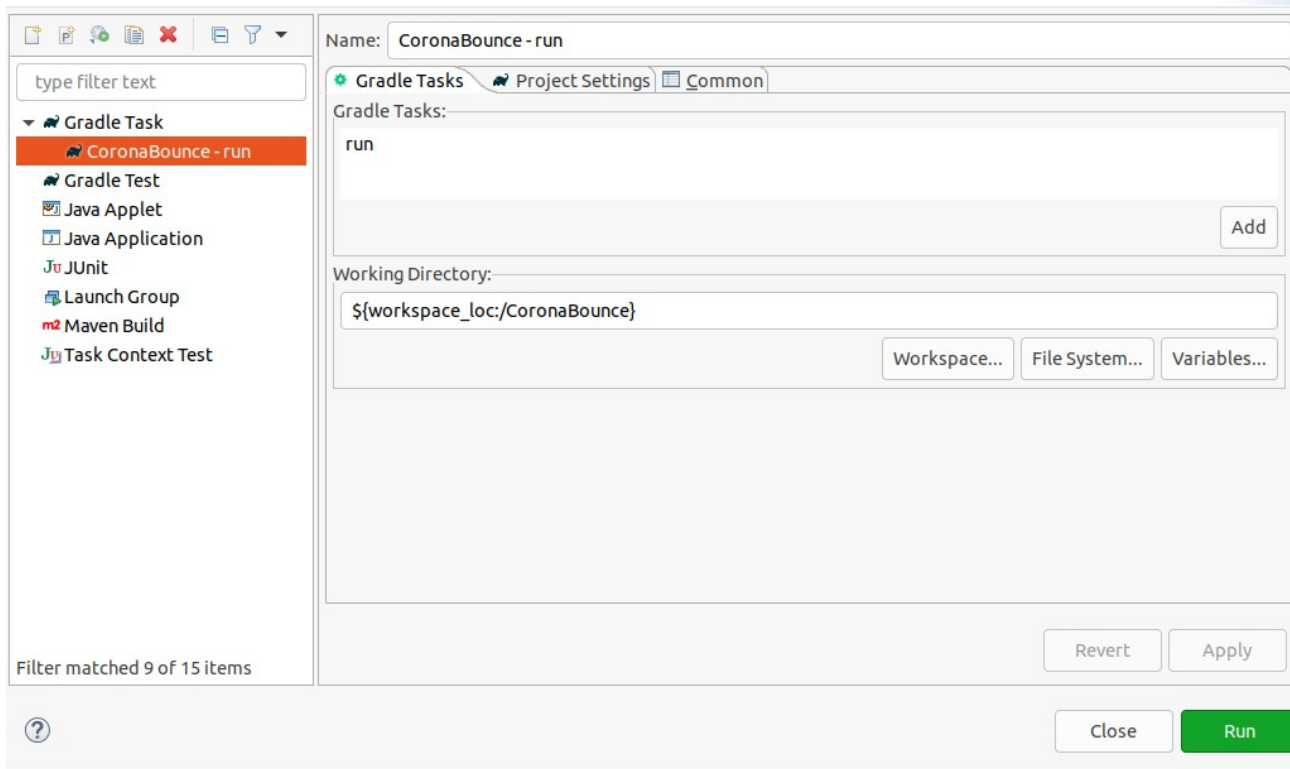
Après l'ouverture d'Eclipse :
File → import
Cette fenêtre apparaîtra



1) Sélectionnez « Existing Gradle Project » → Next ->
Sélectionnez le répertoire du projet → finish



2) Cliquez sur l'icône → run
configuration



3) Sélectionnez CoronaBounce-run → run

Le projet exécutera ensuite.